REAMING POWERPC PERFORMANCE FROM APPLE'S EMULATOR ENGINEER

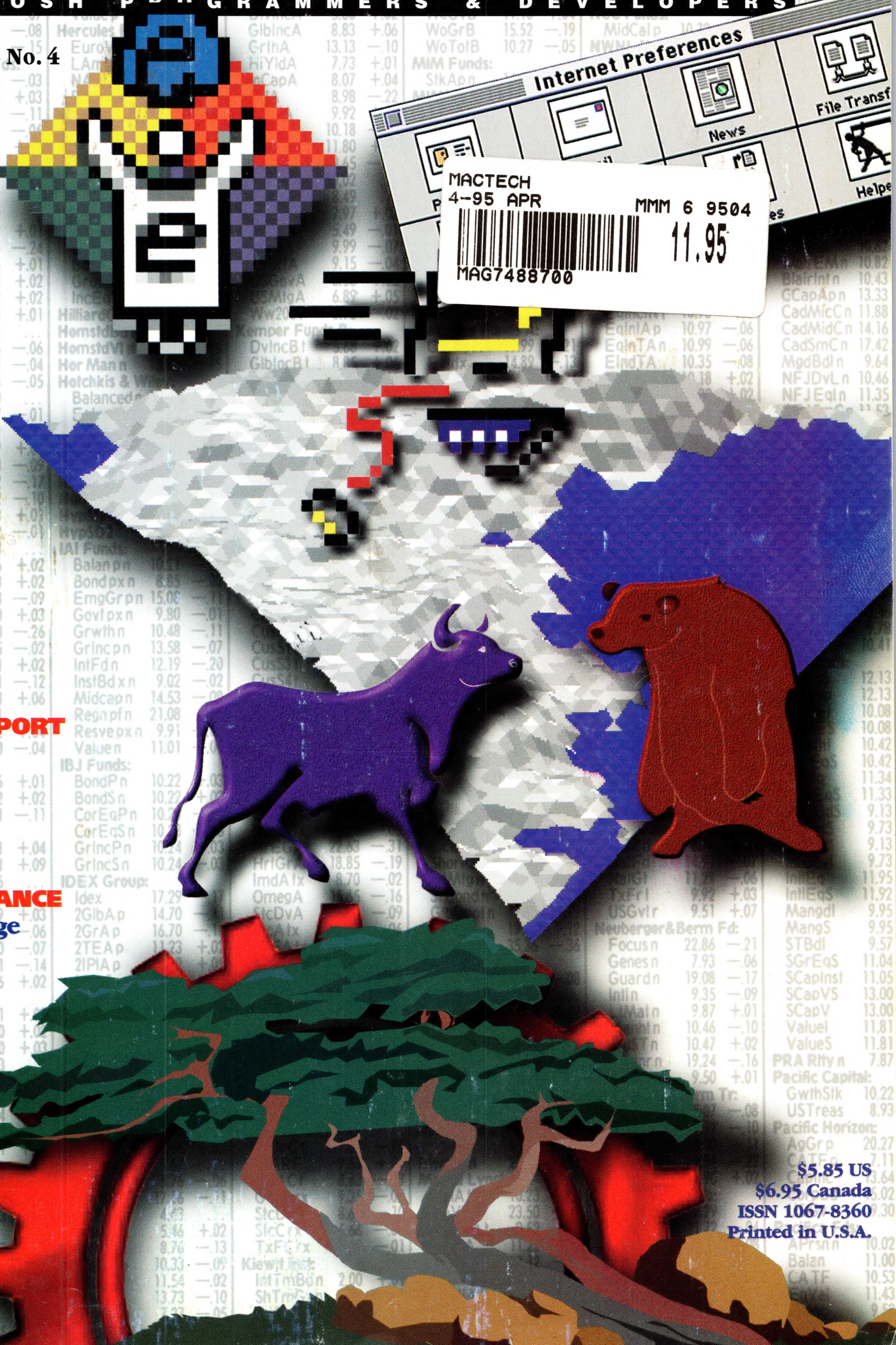
MacTech MAGAZINE™

Formerly MacTutor

FOR MACINTOSH PROGRAMMERS & DEVELOPERS

APRIL 1995 • VOLUME 11, NO. 4

In This Issue!



MACTECH
4-95 APR MMM 6 9504
11.95
MAG7488700

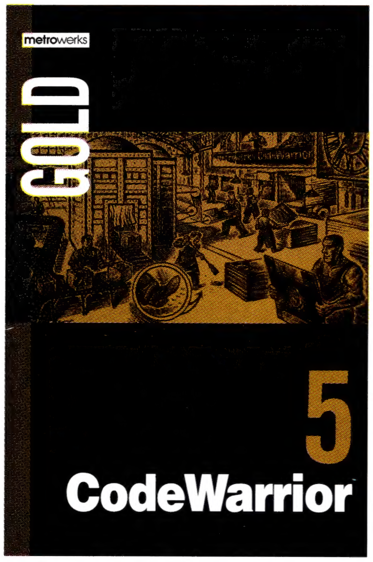
04

32128 74887 8

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

metrowerks® CodeWarrior™

The fastest way to write 68K and Power Macintosh code.



Twelve Compilers!
CodeWarrior Gold includes twelve compilers and CodeWarrior Bronze includes four compilers. With our Power Mac-hosted 68K cross compilers, you can build both 68K and Power Mac apps on the Power Mac. Conversely, with our 68K Mac-hosted Power PC cross compilers you can build 68K and Power Mac apps on a 68K Mac.
Note: CodeWarrior compilers are shipped as fat binaries!

Native IDEs!
The CodeWarrior integrated development environment (IDE) is native on both the 68K Mac and Power Mac. The same easy-to-use environment is used to program in C, C++ and Pascal.
Note: CodeWarrior IDEs are also fat binaries.

Get To Market Faster!
CodeWarrior compilers build your 68K and Power Mac apps at +200,000 lines/minute on a Power Mac 8100, with comparably high rates on less powerful Macs!

Two Free Updates!
CodeWarrior is updated 3 times a year on CD-ROM. Purchase either version of CodeWarrior and you are entitled to receive two future updates free of charge.

New Features!
- •MPW tools included for both the 68k and Power PC
- •C++ Templates supported in the compilers
- •Global optimization improved for the Power PC C/C++ compilers
- •Conditional breakpoints and expression evaluation added for the debugger

Two CodeWarrior versions to choose from!
Choose the CodeWarrior that suits your needs:

	BRONZE 68K Mac	GOLD Power Mac/68K Mac
Power Mac-hosted Power Mac IDE:		
Power Mac-hosted C/C++ generating PPC code		•
Power Mac-hosted Pascal generating PPC code		•
Power Mac-hosted 68K Mac IDE:		
Power Mac-hosted C/C++ generating 68K code	•	•
Power Mac-hosted Pascal generating 68K code	•	•
68K Mac-hosted Power Mac IDE:		
68K Mac-hosted C/C++ generating PPC code		•
68K Mac-hosted Pascal generating PPC code		•
68K Mac-hosted 68K Mac IDE:		
68K Mac-hosted Pascal generating 68K code	•	•
68K Mac-hosted C/C++ generating 68K code	•	•
MPW Tools:		
Power Mac-hosted C/C++ generating PPC code		•
Power Mac-hosted C/C++ generating 68K code	•	•
68K Mac-hosted C/C++ generating PPC code		•
68K Mac-hosted C/C++ generating 68K code	•	•
PowerPlant Application Framework	•	•
PowerPlant Shared Library for Power Mac		•
Source-level debugger for Power Mac		•
Source-level debugger for 68K Mac	•	•
Constructor Visual Interface Editor	•	•
Selected Apple Developer Tools	•	•
30-day money-back guarantee	•	•
3,000 pages of On-line Documentation and much, much more.	•	•

1994 BYTE AWARD OF EXCELLENCE

CodeWarrior 5 Gold
For Power Macintosh & 68K Macintosh development.
LNG 0070..........................$399

CodeWarrior 5 Bronze
For 68K Macintosh development.
LNG 0068..........................$99

Requires: Motorola 68020 or higher or PowerPC 601 processor; 8MB RAM; System 7.1; CD-ROM Drive

metrowerks®

CALL 1-800-377-5416
International (419) 281-1802
Fax (419) 281-6883

Time Is On Your Side

Now VIP-C creates applications that are accelerated for Power Macintosh.

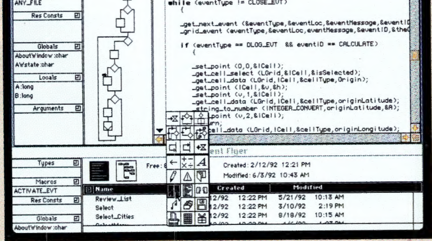
Now, create 68K and Power Macintosh applications faster than ever before. VIP-C's visual design, its application building aids, and its ability to let you modify your program code and interface with immediate feedback, give you a real programming edge. Read further to see why, with VIP-C, time really *is* on your side...

Complete Development System

Now develop with just one comprehensive system. VIP-C integrates all the tools you need to create stand-alone 68K and Power Macintosh applications—right out of the box! Develop applications in a standard language using this intelligent, full-featured Rapid Application Development (RAD) environment. No more moving from tool to tool. No more dependence on non-standard languages.

For All Levels of Programmers

VIP-C offers multiple levels of support to best fit your abilities. If you just want to type C code into its editor, you can—and VIP-C will automatically check the syntax and create a flowchart of your code. For higher-level support, VIP-C features prewritten intelligent prototypes of all the Mac Toolbox calls, high-level VIP-C Functions that simplify low-level Toolbox calls, Resource Editors that let you visually design and create your windows, menus, dialogs, buttons, etc., and even a VIP-C Dispatcher to provide an efficient and reliable application framework—all in one complete development environment.



```
void Calculate (extended A, extended B, extended C, extended D, char Origin[20], char Destination[20])

while (eventType != CLOSE_EVT)
   {

   _get_next_event (&eventType,&eventLoc,&eventMessage,&eventID
   _grid_event (eventType,&eventLoc,eventMessage,&eventID,&theC

   if (eventType == DLOG_EVT  && eventID == CALCULATE)
      {

      _set_point (0,0,&lCell);
      _get_cell_select (LGrid,&lCell,&isSelected);
      _get_cell_data (LGrid,lCell,&cellType,Origin);
      _get_point (lCell,&v,&h);
      _set_point (v,1,&lCell);
      _get_cell_data (LGrid,lCell,&cellType,originLatitude);
      _string_to_number (INTEGER_CONVERT,originLatitude,&A);
      point (v,2,&lCell);
      rn;
      cell_data (LGrid,lCell,&cellType,originLongitude);
```

```
Created: 2/12/92 12:21 PM
Modified: 6/3/92 10:43 AM

Name            Created           Modified
Review_List     12/92  12:22 PM   5/21/92  10:13 AM
Select          12/92  12:22 PM   3/10/92   2:19 PM
Select_Cities   12/92  12:22 PM   8/18/92  10:15 AM
```

User Interface Generation Tools

The VIP-C Dispatcher simplifies program development by automating the main event loop. It acts as a central controller to manage program events by distributing tasks to different routines. User interface items—menus, dialogs, buttons, etc.—are created with integrated resource editors that automatically link user action to your code.

VIP-C 1.5
Everything you need to create complete Macintosh and Power Macintosh applications in one box.

Powerful Prewritten Functions

Functions for the most frequently used features of the Mac interface have been prewritten and stored in a palette for instant use. With the click of a mouse you can lighten your programming workload and speed design and coding time by several orders of magnitude.

On-Line Macintosh Toolbox Calls

VIP-C accesses the complete Mac Toolbox. Search and display prototypes for any function, structure, or macro. No more rooting through multiple volumes of Inside Macintosh!

Attention Database Developers!

VIP-C users now have the power to create commercial quality, multi-user, relational databases using the new VIP Database Manager (sold separately). VIP Database Manager has the built-in functionality of the CXBase Pro database engine (recently chosen by Apple OSA for creating the eWorld content publishing software) giving you an inexpensive way to create professional databases, royalty-free!

Compiler Hotlink

If you have a favorite compiler, use it! Compile and run your application directly from a menu item in the VIP-C editor. VIP-C supports: THINK C, MPW-C, and CodeWarrior (each sold separately).

Buy VIP-C now for a Limited-Time Special Price of only $295. Act now and get VIP-C Database Manager for only $149.

VIP-C's suggested retail price is $495. But from now until June 1, 1995, you can order VIP-C for $295. Call Mainstay to order today.

Mainstay

(800) 484-9817 Ext. 8472

591-A Constitution Ave.	71 rue des Atrébates
Camarillo, CA 93012	B-1040 Brussels, Belgium
fax: (805) 484-9428	32-2/733.97.91

MacTech MAGAZINE Formerly MacTutor
FOR MACINTOSH PROGRAMMERS & DEVELOPERS
A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?** If you have any questions, feel free to call us at 310/575-4343 or fax us at 310/575-0925.

DEPARTMENTS	Internet	eWorld	CompuServe	AppleLink	America Online
Orders, Circulation, & Customer Service	custservice@xplain.com	MT.CustSvc	71333,1063	MT.CUSTSVC	MT CUSTSVC
Editorial	editorial@xplain.com	MT.Editors	71333,1065	MT.EDITORIAL	MT EDITORS
Programmer's Challenge	progchallenge@xplain.com	MT.PrgChal	71552,174	MT.PROGCHAL	MT PRGCHAL
Ad Sales	adsales@xplain.com	MT.AdSales	71552,172	MT.ADSALES	MT ADSALES
Online Support	online@xplain.com	MT.Online	–	–	–
Accounting	accounting@xplain.com	accounting@xplain.com	–	–	
Marketing	marketing@xplain.com	MT.Mktg	–	–	–
Press Releases	pressreleases@xplain.com	pressreleases@xplain.com	–	–	–
General	info@xplain.com	info@xplain.com	71333,1064	MACTECHMAG	MacTechMag
Online support area	ftp://ftp.netcom.com/pub/xp/xplain	*use shortcut:* MACTECH	*type* GO MACTECHMAG	*see* Third Parties: Third Parties (H-O)	*use keyword:* MACTECHMAG

Authors & Regular Contributors

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology. We are dedicated to the distribution of useful programming information without regard to Apple's developer status. For information on submitting articles, ask us for our **writer's kit** which includes the terms and conditions upon which we publish articles.

Richard Clark
General Magic
Mountain View, CA

David R. Mark
M/MAC
Arlington, VA

Mike Scanlin
Programmer's Challenge
Mountain View, CA

Chris Espinosa
Apple Computer, Inc.
Cupertino, CA

Jordan Mattson
Apple Computer, Inc.
Cupertino, CA

Symantec Technical Support Group
THINK Division
Eugene, OR

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 1617 Pontius Avenue, 2nd Floor, Los Angeles, CA 90025-9555. Voice: 310/575-4343, FAX: 310/575-0925. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Second Class postage is paid at Los Angeles, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 250055, Los Angeles, CA 90025-9555.

TABLE OF CONTENTS

April 1995 • Volume 11, No. 4

MacTech MAGAZINE™ Formerly MacTutor
FOR MACINTOSH PROGRAMMERS & DEVELOPERS

THE EDITOR'S VIEWPOINT

By Scott T Boyd, Editor

It's a Flat World Out On The Web

This month I wanted to talk about two things – the World Wide Web, and OpenDoc. While mulling it over, I ran across a Usenet posting by Mark Hanrek of The Information Workshop, and decided to let him have at it. Here's his guest editorial. We welcome your comments at editorial@xplain.com – Ed stb

And Now Here's Mark...

Every time I communicate about the aspects of the WWW standard that are deeply flawed, I look for existing examples of things done right to prove my point. And every time, OpenDoc comes up.

It will be great if the OpenDoc work is having any influence at all on those who are in control of the Web page standard, because the current WWW path is really quite unfortunate for us all.

I realize that the Web page standard started life as an ad-hoc thing that turned out to be useful, it caught on, and then spread. But now that the rest of the world has grabbed onto it, there really needs to be the courage to stop and set things straight, and to do it properly.

The Fatal Flaw

The Web page people seem to have their heads in a 'computer printout' and 'linear list' space. The term 'inline graphics' is revealing. I get the feeling they thought it was cool to leave out the specification for the absolute position of objects on a page. They 'leave it up to the interpretation of the particular browser'. On the Macintosh alone, one gets three completely different interpretations using Netscape, Mosaic, and MacWeb respectively.

That's bad. This is a fatal flaw. And unfortunately, one we will have to live with somehow.

This is the reason no one can create a WYSIWYG HTML editor on the Mac, even though there are umpteen tools on the Mac one can use to author one in short order, such as HyperCard, SuperCard, Prograph, ClarisWorks HTML XTND, class libraries used with Symantec and MetroWerks environments, and eventually, OpenDoc.

Attempting to author an editor puts one in the position of having to take a stand on which is the right way to display things. A support mess. Lots of dissatisfied people. Why bother!

As a result, everyone relents, and has little choice but to rally around one browser (Netscape). This kills competition, and, well, lots of other essential elements.

What's Happening Here

Web pages are for *display*, and must be *interactive*, regardless of what anyone says. If anyone were to go out and design a cross-platform standard for the *display* of *interactive* multi-media, and do it in a manner that stood on the shoulders of what we have already learned, addressed itself to the unknown needs of the future, and accommodated the commercial realities that motivated it in the first place......they would be designing and creating something like OpenDoc, not HTML.

When one takes the HTML standard, and performs a comparison against OpenDoc, well, it's quite shocking. HTML starts us back at the beginning of computer time.

So much wasted energy... So many professionals using their slick WYSIWYG editors on their GUI workstations, struggling with the equivalent of editing Postscript directly to try and get results that look halfway decent after surviving the non-deterministic 'interpretations' of Netscape, Mosaic, and other Web browsers.

[shaking head] What a sad mess. I am embarrassed to be associated with it. And powerless of influence it. If you try to tell them anything, then 'you just don't understand'. I can't help but feel the SGML people have their vise grip on things 'cuz it'll be their last hurrah. :)

You're dang right I'm bummed. But there is hope. With OpenDoc illuminating a more appropriate way, perhaps life will take one of its funny twists and turns sometime in the near future.

Go Team!

Thankfully, OpenDoc exists, and is one of the most inspired, well-conceived, accommodating, courageous, and honestly handled American technology developments I have ever witnessed in my twenty three years of computer design and engineering.

I say all of this as a vote of confidence, a show of respect, and as encouragement to the OpenDoc team.

Well done.

– *Mark Hanrek, The Information Workshop, hanrek@cts.com*

The Information Workshop http://www.electriciti.com/~infoman/
SuperCard Home Page http://www.electriciti.com/~supercrd/

Sorry You Missed Woodstock?

Get your early reservation in *now* for MacHack X (June 22-24) and save yourself some dough. Line up some hacking buddies, and get ready for The Best Hack Contest (sponsored by yours truly) – it's a great way to be a part of the 10th Annual MacHack experience. e-mail expotech@aol.com, check out http://www.hax.com/HackContest and call Expotech at (313) 882-6942.

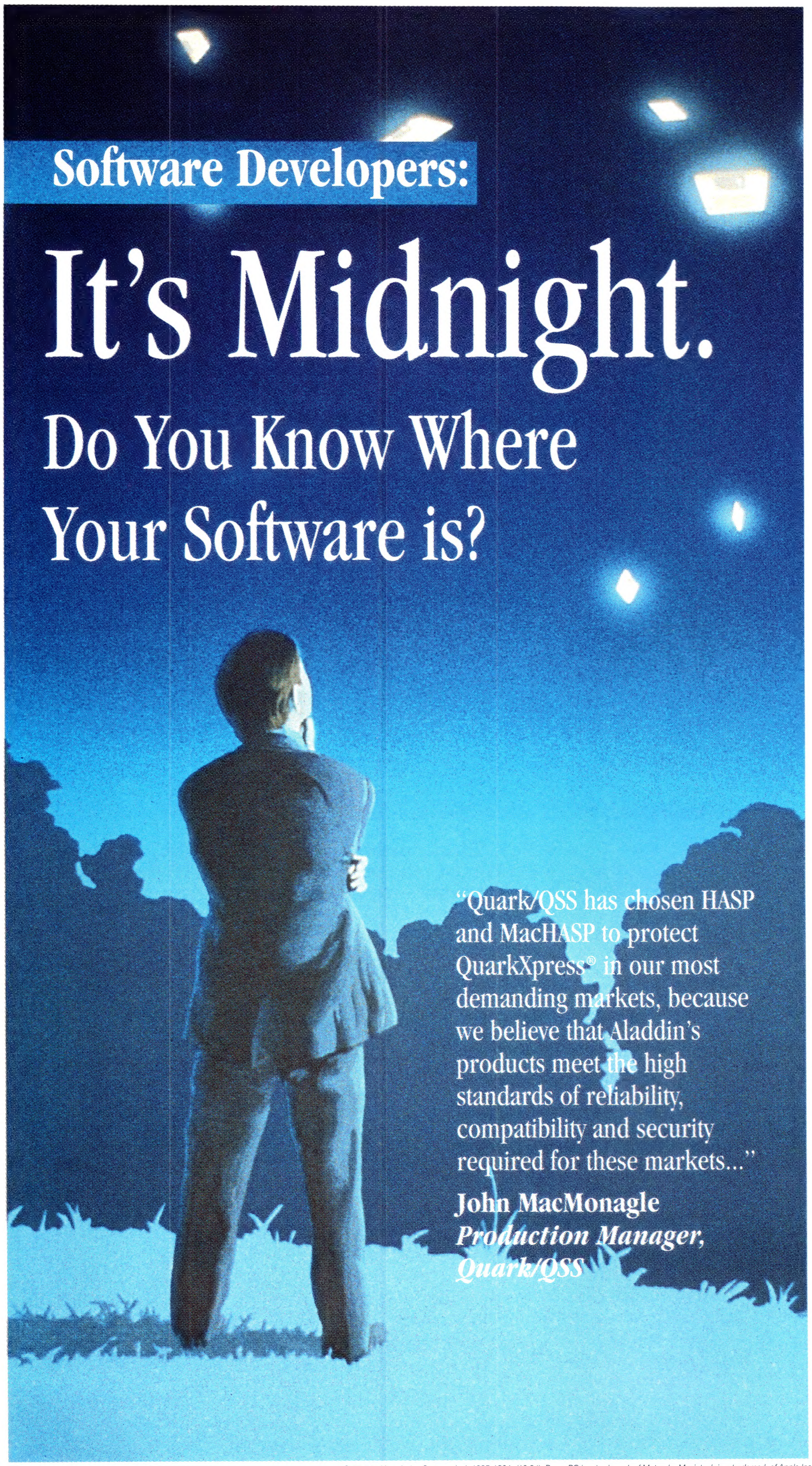
"If you stay awake the whole time, it's only 8 cents per minute!"
– *Keith Stattenfield, someone who* ***never*** *sleeps at MacHack*

"… they give good prizes… :)"
– *Doug McKenna, Mathemaesthetics*

Food For Thought

Macintosh: Changing the world, one person at a time! (I wish they'd hurry up!) – *Chris De Salvo, phixus@deltanet.com*

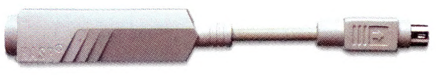
Does anyone in the Fed understand the concept of time-delayed feedback loops? – *Jorg 'jbx' Brown*



"Quark/QSS has chosen HASP and MacHASP to protect QuarkXpress® in our most demanding markets, because we believe that Aladdin's products meet the high standards of reliability, compatibility and security required for these markets..."

John MacMonagle
Production Manager, Quark/QSS

Bringing software into the world is a little like bringing up children. You always know where they start, but you seldom know where they'll end up. These days, with illegal use of software so common, concerned developers have good reason to worry about the products of their labor. That's where MacHASP comes in.



Like a responsible babysitter, MacHASP accompanies your software wherever it goes. With MacHASP there, your software won't run out of control. Without MacHASP, in fact, your software won't run at all.

For developers, MacHASP provides the highest level of security and reliability. For legitimate users, MacHASP is a friendly and transparent solution. Once connected, they won't even feel it's there.

And if your child wants to play with its friends, a single Net-MacHASP lets it run free around a local area network. But always under your supervision and control.

Get serious about software protection. Since 1984, nearly one million HASP keys have enabled thousands of PC & Mac software developers, in more than 60 countries, to protect their software. To find out why MacHASP is considered the best product in the market, order your MacHASP Developer's Kit today.

1-800-223-4277

ALADDIN

The Professional's Choice

North America	**Aladdin Software Security Inc.** Tel: (800) 223 4277, 212-564 5678 Fax: 212-564 3377 E-mail: sales@hasp.com
Intl Office	**Aladdin Knowledge Systems Ltd.** Tel: 972-3-537 5795, Fax: 972-3-537 5796 AppleLink: ALADDIN.KNOW E-mail: aladdin@aladdin.co.il
United Kingdom	**Aladdin Knowledge Systems UK Ltd.** Tel: 0753-622266, Fax: 0753-622262
France	**Aladdin France SA** Tel: 1 40 85 98 85, Fax: 1 41 21 90 56

member of spa

■ **Australia** Conlab 3 8985685 ■ **Benelux** Aladdin Benelux 080 782098 ■ **Czech** Atlas 2 766085 ■ **Chile** Micrologica 2 222 1388 ■ **Denmark** Berendsen 39 577100 ■ **Egypt** Zeineldein 2 3604632 ■ **Finland** ID-Systems 0 870 3520 ■ **Germany** CSS 201 278804 ■ **Greece** Unibrain 1 6856320 ■ **Italy** Partner Data 2 26147380 ■ **Japan** Athena 3 58 213284 ■ **Korea** Dae-A 2 848 4481 ■ **Mexico** SiSoft 5 5439770 ■ **New Zealand** Training 4 5666014 ■ **Poland** Systherm 61 480273 ■ **Portugal** Futurmatica 1 4116269 ■ **South Africa** D Le Roux, 11 886 4704 ■ **Spain** PC Hardware, 3 4493193 ■ **Switzerland** Opag 61 7169222 ■ **Taiwan** Teco 2 555 9676 ■ **Turkey** Mikrobeta 312 467 7504

Eddy Award Winner *for* ***Best New Developer Tool***
– MacUser Editors Choice Awards, 1993

"A distinct improvement over ResEdit."
– MacTech / MacTutor

"Resorcerer's data template system is amazing!"
– Bill Goodman, author of Compact Pro

"Nuke ResEdit! Resorcerer is mission-critical for us."
– Dave Winer, Userland Frontier

"The color pixel editors are wonderful! A work of art!"
– Dave Winzler, author of Microseeds Redux

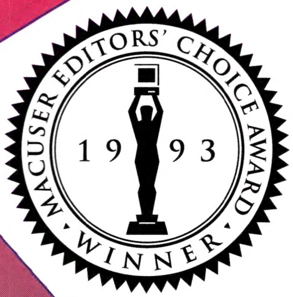
"Every Macintosh developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

"…a wealth of time-saving tools."
MacUser Review, Dec. 1992



RESORCERER®

The Resource Editor for the Macintosh Wizard

Version 1.2.4

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: $256 (decimal)
(Educational, quantity, or other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red,
International Shipping

Downloadable Demos/Updaters:
AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

New 1.2 Features:

- **New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors**
- **Powerful icon family editing (all 9 icon types)**
- **Color pixel anti-aliasing, dithering, and lots more**
- **Complete 'PICT' disassembly and reassembly**
- **Resource sorting; ROM resource browsing**
- **120 template field parsing types now supported**
- **New insertion & deletion template field types**
- **Text-only 'PICT' resources**
- **Lots of improvements throughout**

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.
P.O. Box 298 • Boulder • CO • 80306-0298 • USA
Phone: (303) 440-0707 • Fax: (303) 440-0504
AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com

By Dave Mark, MacTech Magazine Regular Contributing Author

Sprocket and the Drag Manager

Adding an important technology to Sprocket

This month's column uses Sprocket to take the Drag Manager out for a quick spin.

I know we were originally scheduled to talk about Sprocket's menuing model, but we just couldn't bring everything together in time. We'll definitely get to it next month.

If you've never used the Drag Manager before, bring up the Scrapbook and scroll over to your favorite PICT. Now click on the PICT and drag it out of the Scrapbook window. If you have the Drag Manager installed on your Mac (more on that in a second), you'll notice a grey rectangle (representing the PICT) following your mouse. Drag the rectangle over any open Finder window. Each time your mouse enters a window, a hilighting rectangle is drawn, telling you that this window is the **focus** of this drag. If you move the mouse outside the window, the focus rectangle disappears. This focus lets you know which window is to receive the object you are dragging around. If you release the mouse button inside one of your Finder windows (or on the desktop), a **clipping file** will be created. The clipping file contains the picture you just dragged from the Scrapbook. If you double-click on a clipping file, the Finder displays its contents in a window. If you drag the clipping file around, it acts just like its contents. That is, if you drag a PICT clipping file, it is exactly as if you were dragging the PICT contained in the file. Try it. Drag a Scrapbook PICT onto the desktop, then drag the resulting clipping file back into the Scrapbook.

About the Drag Manager

Under System 7.5 or later, the Drag Manager is built right in. Earlier systems required the installation of the Dragging Enabler extension or its replacement, the Macintosh Drag and Drop extension (version 1.1 is the latest). No matter what System you have, you'll also need the Clipping Extension if you want the **Finder** to work with clipping files.

Apple has released a Drag Manager SDK, which contains everything you'll need to work with the Drag Manager, including some great sample code and two critical DocViewer documents. The first of these, the *Drag Manager Programmer's Guide* is basically an *Inside Macintosh* chapter on the Drag Manager. The second document, *Drag and Drop H.I. Guidelines*, tells you how the Drag Manager is supposed to look to the user.

“In the near future, the user will expect Drag and Drop support and will likely be unforgiving if it is absent.”

If you've never worked with the Drag Manager before, there are two excellent articles that will help ease you into the Drag Manager way of thinking. The first of these, was written by Steve Kiene and appears in the June 1994 issue of *MacTech Magazine* (write to custservice@xplain.com to see if you can order the back issue). The article is called “Macintosh Drag and Drop” and it takes you through the basic Drag Manager concepts and terminology. The second article, “Drag and Drop From the Finder” by Dave Evans and Greg Robbins appeared in the December 1993 issue of **develop** (issue 16). This article has a slightly different slant than the *MacTech* article. If you can, read them both before you dive into the *Programmer's Guide*. You might also want to check out the article “Implementing Elegant Drag & Drop for

Styled Text Fields", by David Simmons, in the November 1994 issue of *MacTech*. The article is based on SmallTalk, but is very readable even if you don't speak the language.

This Month's Program

This month's program provides a basic demonstration of the Drag Manager. First, we'll create a new class called TPictureWindow. A TPictureWindow is a simple, non-growable, non-zoomable, non-scrollable window with a PICT centered in it. The TPictureWindow supports dragging in both directions. That is, you can drag a picture into a TPictureWindow and you can drag a picture from a TPictureWindow as well.

Once our TPictureWindow class is added to our project, we'll modify our code so that a TPictureWindow is the default document type instead of a TDocWindow. This means that when you select New from the File menu, you'll create a TPictureWindow instead of a TDocWindow.

Important note: We've changed our default Sprocket project from SprocketSample to SprocketStarter. From now on, all of our Sprocket-based programs will start from the latest baseline of SprocketStarter and add the resources, classes, and changes that relate to that month's topic.

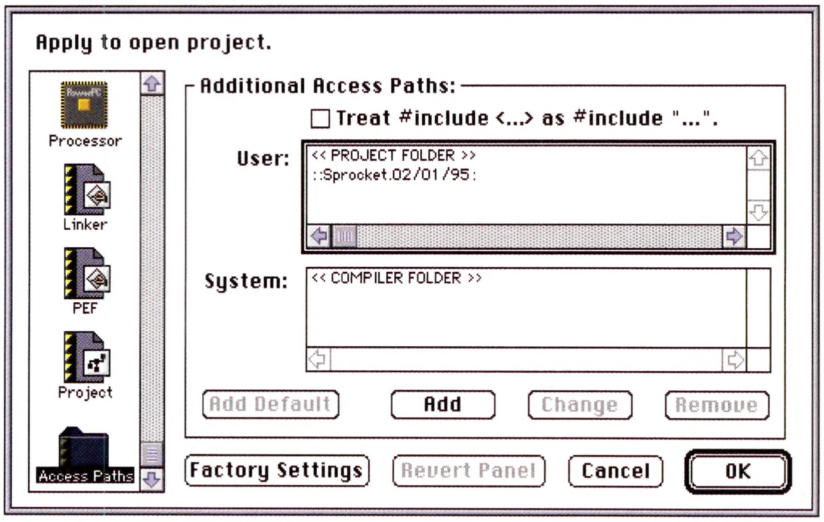
SprocketDragger

This month's program is based on the files in the folder "SprocketStarter,95.02.01". In order to distinguish it from the original, I renamed the modified SprocketStarter folder to be "SprocketDragger,95.02.01". That's the only reference to SprocketDragger you'll see. To keep things as clear and as simple as possible, the file names were not changed when I went from SprocketStarter to SprocketDragger. This way, when you want to compare the old and new versions of SetupApplication(), for example, you'll be comparing two different versions of SprocketStarter.cp.

Both "SprocketStarter,95.02.01" and "SprocketDragger,95.02.01" are based on the Sprocket files found in "Sprocket,95.02.01". Since Symantec still hasn't released a product that uses the new Universal Headers, all of this code was built using CodeWarrior CW5.

Building and Running SprocketDragger

Depending on whether you have a PowerMac or not, launch either SprocketStarter.68K.µ or SprocketStarterPPC.µ. You'll need CodeWarrior CW5 (at least) to build either project. Two likely places where you might run into trouble: the Language and AccessPaths preference panes. Select Preferences from the Edit menu. Scroll to and click on the Access Paths icon. Be sure that an access path leading to your latest Sprocket folder is listed in the User: area (Figure 1).


Apply to open project.

Additional Access Paths:

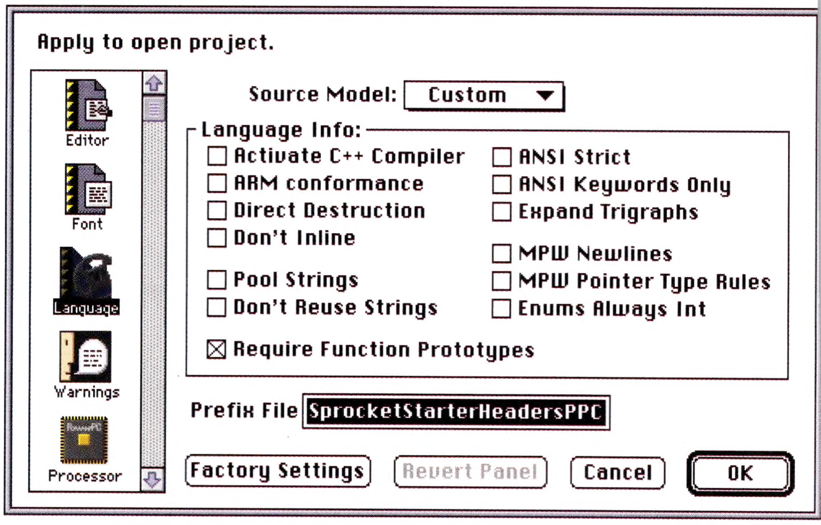
☐ Treat #include <...> as #include "...".

User:
<< PROJECT FOLDER >>
::Sprocket.02/01/95:

System:
<< COMPILER FOLDER >>

Figure 1. Be sure a path to your Sprocket folder is added to the User: area.

Next, scroll to and click on the Language icon. Check to be sure that the file listed in the Prefix File field goes with the ".pch++" in your project (Figure 2). For example, if the precompiled header source file SprocketStarterHeadersPPC.pch++ is included in your project, CodeWarrior will build a file called SprocketStarterHeadersPPC. By specifying that file in the Language pane, you are asking CodeWarrior to include that precompiled header in each file it compiles. Why precompile? Since the header is mostly #include files, which likely won't change during your development cycle, compiling them in advance will definitely speed up your build time. Take a minute to double-click on the ".pch++" file included in your project.


Apply to open project.

Source Model: Custom

Language Info:

☐ Activate C++ Compiler
☐ ARM conformance
☐ Direct Destruction
☐ Don't Inline
☐ Pool Strings
☐ Don't Reuse Strings
☐ ANSI Strict
☐ ANSI Keywords Only
☐ Expand Trigraphs
☐ MPW Newlines
☐ MPW Pointer Type Rules
☐ Enums Always Int
☒ Require Function Prototypes

Prefix File SprocketStarterHeadersPPC

Figure 2. Be sure that the Prefix Header field points to the right precompiled header for your machine.

There are Billions of reasons to protect your software.

You only need one reason to protect with Sentinel:

Piracy is the greatest threat to the world's software industry. Developers lose billions in sales to software piracy each year. Protect your software and get all the revenue you deserve.

It's the worldwide standard in software protection.

More developers rely on Sentinel®, from Rainbow, than any other software protection in the world.

And for good reason. Sentinel performs where it matters most: leading the industry in technology, quality, reliability and support.

So when it's time to protect your Macintosh application - and you need just-in-time delivery - protect with Sentinel. Protect with confidence.

Call Rainbow for a Sentinel Developer's Kit and a FREE copy of *"The Sentinel Guide to Securing Software."*

Mac™ OS

ISO 9002 CERTIFIED

Call Today!
1-800-852-8569

RAINBOW TECHNOLOGIES

SENTINEL
Securing the future of software

9292 Jeronimo Road • Irvine, CA 92718 • Tel: 714/454-2100 • Fax: 714/454-8557 • *Applelink D3058* •*International offices:* U.K. (44) 1932 570066 • France (33) 1 47 38 21 21 • Germany (49) 89 32 17 98 0 • North Carolina 800/843-0413

By the way, if you get 3 link warnings concerning a possible duplicate 'ckid' resource, just ignore them. You're seeing a CodeWarrior bug that comes up when you include your resource file in the project file. The bug won't affect program execution.

Once CodeWarrior finishes compiling, it will run SprocketDragger. A splash screen will appear (however briefly) and a picture window will appear (Figure 3). An empty floating window will also appear, but since we don't use that window, just ignore it. Better yet, can you figure out how to get rid of the floating window entirely? Hint: It is an object of class TToolWindow and it is created in the file SprocketStarter.cp.

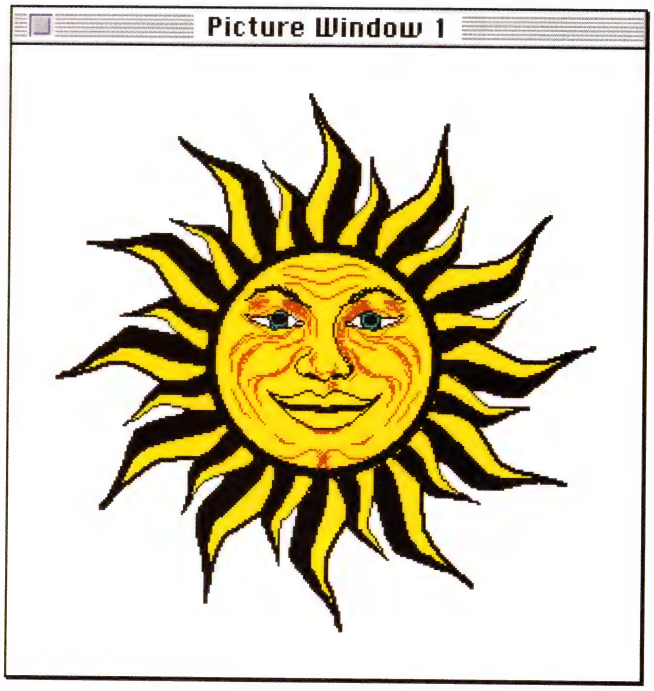


Figure 3. The TPictureWindow *that appears when* SprocketDragger *starts up.*

The picture window displays the default picture defined by the TPictureWindow class. Basically, this picture appears when a TPictureWindow has not had a PICT dragged to it yet. With SprocketDragger still running, go to the Finder and drag an icon from the Finder onto the TPictureWindow. The TPictureWindow does not react to a non-PICT drag.

Now bring up the Scrapbook and drag a picture from the Scrapbook onto the picture window. This time, a highlighting rectangle appears around the border of the window, indicating that the window is willing to accept this data. If you release the drag inside the picture window, the picture you just dragged will appear in the window.

If you have the Clipping extension installed, try dragging the picture in the picture window into a Finder window. My favorite is to drag from the picture window into the trash. No matter what Finder window you select, a clipping file will appear containing the dragged PICT. Try dragging an image from the Scrapbook to the desktop to create a clipping file, then drag the clipping file onto the picture window. Once again, the highlighting rectangle appears in the picture window as the focus of your drag enters the window. This indicates that the picture window is accepting the drag. And why not, since the clipping file looks just like a PICT to the picture window.

Try dragging a picture from the picture window into a window that can't receive a drag. For example, try running Microsoft Word or any other non-drag-friendly application and drag from the picture window into the Word window. When you release the mouse, the dragging outline will zoom back to the original window, indicating that the drag was rejected.

Next, select **New** from the **File** menu to create a second picture window. When the window appears, drag it to the side so you can see both picture windows at the same time. Notice that the windows are numbered so you can tell them apart. Assuming you now have a different picture in each window, try dragging from one window to the other.

There are a lot of other ways you can test out your new dragging environment. For now, though, let's get into the resources and source code that makes this all possible.

SprocketDragger Resources

SprocketDragger added two new resources to the file SprocketStarter.rsrc. A PICT with a resource ID of 1025 was added to give us a default picture to display in windows that hadn't received a drag yet. A WIND with a resource ID of 1028 acted as a template for the TPictureWindow window.

Modify SprocketStarter.cp

The first source code change I made was in the file SprocketStarter.cp. I modified the routine CreateNewDocument() to create a TPictureWindow instead of a TDocWindow:

```
OSErr
CreateNewDocument(void)
  {
  TPictureWindow  *aNewWindow = new TPictureWindow();

  if (aNewWindow)
    return noErr;
  else
    return memFullErr;
  }
```

Remember, Sprocket calls CreateNewDocument() whenever an 'oapp' Apple event is received or when **New** is selected from the **File** menu. Since we haven't gotten into Sprocket's menu model yet, we didn't implement a separate, TPictureWindow test menu.

Add the TPictureWindow Class

Once those changes were made, the only other thing left to do was to create the TPictureWindow class and add it to the project. As you'd expect, the TPictureWindow class was

PROGRAPH CPX. TAKE A MAGIC CD-ROM RIDE.

Build Applications As Fast As You Can Visualize Them

Yes, *visualize* them. No ifs, ands or C++.

There's real magic in Prograph CPX's pictorial programming environment. Not just another pretty interface, its visual magic includes the actual down and dirty of programming itself. What's the payoff? Maximum productivity.

Prograph CPX is the award-winning OOP tool of choice for client/server solutions, sophisticated multimedia authoring and all your development needs. And you won't have to charm any snakes to deliver killer applications in less time and with less money.

Rapid application development.
Maximum flexibility.
And the ride of your life.

Test drive Prograph absolutely free and experience why it's the #1 rapid application development tool.

Just call 800.927.4847.

Ask for your magic CD-ROM ride.

prograph international

P.O. Box 2206, San Francisco, CA 94126-2206
Internet: info@prograph.com AOL: PrographUS

implemented in the files PictureWindow.cp and PictureWindow.h. As we go through the source code in these two files, you'll see how we took advantage of the Drag Manager code already built into Sprocket.

PictureWindow.h

PictureWindow.h starts off in the usual way, by defining a constant to prevent an infinite include loop and by including the declarations associated with its base class, TWindow.

```
#ifndef     _PICTUREWINDOW_
#define     _PICTUREWINDOW_

#ifndef     _WINDOW_
#include    "Window.h"
#endif
```

The TWindow class implements all the basic features that make up a general window object. The TDocWindow class (the one with the spinning cursor which we worked with last month) is based on the TWindow class. The TPictureWindow class borrows heavily from the design of the TDocWindow class and is also based on the TWindow class.

```
class  TPictureWindow : public TWindow
{
  public:
                 TPictureWindow();
  virtual        ~TPictureWindow();

  virtual WindowPtr    MakeNewWindow( WindowPtr behindWindow );

  virtual void       Draw(void);

  virtual void       ClickAndDrag( EventRecord *eventPtr );

  virtual  OSErr     DragEnterWindow( DragReference dragRef );
  virtual  OSErr     DragInWindow( DragReference dragRef );
  virtual  OSErr     DragLeaveWindow( DragReference dragRef );
  virtual  OSErr     HandleDrop( DragReference dragRef );

// Non-TWindow methods...
  virtual PicHandle LoadDefaultPicture();
  virtual void       CenterPict( PicHandle picture,
                             Rect *destRectPtr );
  virtual Boolean    IsPictFlavorAvailable(
                               DragReference dragRef );
  virtual Boolean    IsMouseInContentRgn(
                               DragReference dragRef );
```

The TPictureWindow class uses 4 data members. fgWindowTitleCount is a static member, which means that a single unsigned long is shared among all TPictureWindow objects. fgWindowTitleCount is incremented every time you create a new TPictureWindow and provides the number you see in the TPictureWindow title bar.

```
protected:
  static unsigned long fgWindowTitleCount;
```

fCanAcceptDrag gets set to true if the current drag contains data that is acceptable to the TPictureWindow (in other words, PICT data). fDraggedPicHandle starts off as nil but points to a PICT if one is dragged into the window. fIsWindowHighlighted is true if the window's drag highlighting is currently turned on.

```
  Boolean          fCanAcceptDrag;
  PicHandle        fDraggedPicHandle;
  Boolean          fIsWindowHighlighted;
};

#endif
```

PictureWindow.cp

PictureWindow.cp starts off with a pair of constants that define the resource IDs for the WIND template and the default PICT.

```
const short     kPictureWindowTemplateID = 1028;
const short     kDefaultPICTResID = 1025;
```

"PictureWindow.h" contains the TPictureWindow class declaration. <ToolUtils.h> contains the declaration of the routines NumToString() and GetPicture().

```
#include "PictureWindow.h"
#include <ToolUtils.h>
```

Before we get into the TPictureWindow methods, we'll initialize the static that tracks the index used in the window titles. We can't initialize the static inside a method, becuase the initialization would be redone every time a new TPictureWindow was created (or the initializing method got called).

```
unsigned long  TPictureWindow::fgWindowTitleCount = 0;
```

The TPictureWindow constructor first initializes fDraggedPicHandle (since we don't have a dragged-in picture yet). It then bumps fgWindowTitleCount so our first window starts with the number 1 instead of 0. Finally, it calls the inherited CreateWindow() to create a new window. CreateWindow() belongs to the TWindow class and calls the MakeNewWindow() method, which we have overridden (ours is right below the destructor).

```
TPictureWindow::TPictureWindow()
{
  fDraggedPicHandle = nil;

  TPictureWindow::fgWindowTitleCount++;
  this->CreateWindow();
}
```

The destructor doesn't do anything. If we intended SprocketDragger to be a "real" application, we'd free up the memory allocated to any received picture, assuming fDraggedPicHandle was not nil.

```
TPictureWindow::~TPictureWindow()
{
}
```

MakeNewWindow() creates a new window based on our picture window WIND template. Basically, this method was lifted from the TDocWindow class.

```
WindowPtr
TPictureWindow::MakeNewWindow( WindowPtr behindWindow )
```

DELL OFFERS MACINTOSH DEVELOPERS THE CHANCE TO OPEN UP A NEW WINDOW.

OPTIPLEX™

THE DELL®
DEVELOPER SYSTEM

DELL® OPTIPLEX™ 590/L
90MHz PENTIUM PROCESSOR

- 528MB Hard Drive/24MB RAM
- 1 MB VRAM
- Microsoft Mouse
- Sound Blaster 16 with Triple Speed CD-ROM & Speakers
- Dell Ultrascan 15" Trinitron
- 3Com EtherLink III Combo
- 3-year limited warranty*
- Microsoft Windows NT Workstation V3.5
- Microsoft MSDN Level II Membership
- Microsoft Visual C++ Subscription

$3,999
Product Code **#300622**

The Dell Developer system will set you up with everything you need to start developing great applications for Windows® 95. With 90 Mhz Pentium® power and speed, Plug-n-Play, advanced power management and much more.

Included is the Microsoft Developer Network Level II subscription which gives you all the information and SDK's that you need for the whole year.

Also included is the Microsoft® Visual C++™ Subscription which gives you the tools you need for a whole year.

With these subscriptions you will get the Windows 95 SDK and Windows 95 specific tools, when they are released to developers.

DELL®

(800) 627-9862 MONDAY-FRIDAY 7AM-9PM CT • SATURDAY 10AM-6PM CT
SUNDAY 12PM-5PM CT • CANADA CALL 800 668-3021

```
{
  WindowPtr  aWindow;
  Str255   titleString;
  GrafPtr    savedPort;

  GetPort(&savedPort);

  aWindow = GetNewColorOrBlackAndWhiteWindow(
                             kPictureWindowTemplateID,
                             nil, behindWindow );

  if (aWindow)
  {
    GetWTitle(aWindow,titleString);
    if (StrLength(titleString) != 0)
    {
      Str255 numberString;

      NumToString( fgWindowTitleCount, numberString );

BlockMove(&numberString[1],&titleString[titleString[0]+1],
                                         numberString[0]);
      titleString[0] += numberString[0];
    }
    SetWTitle(aWindow,titleString);

    SetPort(aWindow);

    ShowWindow(aWindow);
  }
  SetPort(savedPort);

  return aWindow;
}
```

The Draw method gets called in response to an update event. `Draw()` checks to see if we have a dragged in PICT. If so, we draw it, otherwise we draw the default picture. `CenterPict()` is an old centering routine I stole from volume I of the Primer.

```
void
TPictureWindow::Draw(void)
{
  PicHandle  pic;
  Rect   r;

  r = fWindow->portRect;
  EraseRect( &r );

  if ( fDraggedPicHandle == nil )
    pic = this->LoadDefaultPicture();
  else
    pic = fDraggedPicHandle;

  this->CenterPict( pic, &r );
  DrawPicture( pic, &r );
}
```

`ClickAndDrag()` gets called when a click occurs in the window's content region and is immediately followed by a drag. This means that the user is trying to drag a picture from our window. The drag might go outside our application, to another part of our application, or even to another part of our window. We'll start a new drag by calling `NewDrag()`.

```
void
TPictureWindow::ClickAndDrag( EventRecord *eventPtr )
{
  OSErr          err;
  DragReference  dragRef;
  RgnHandle      dragRegion, tempRgn;
  Rect           itemBounds;
  Handle         flavorDataHandle;
```

```
   err = NewDrag( &dragRef );
   if ( err != noErr )
    return;

  if ( fDraggedPicHandle == nil )
    flavorDataHandle = (Handle)this->LoadDefaultPicture();
  else
    flavorDataHandle = (Handle)fDraggedPicHandle;

  HLock( flavorDataHandle );
```

We'll then tell the Drag Manager that this drag contains PICT data. Think of flavors as types, such as those found in the scrap. The second parameter is a serial number we want to assign to this item. We'll use the `WindowPtr` as a serial number, though we could have used the number 1L or even 599923L. When you get into drags containing multiple items, this becomes more of an issue.

```
  err = AddDragItemFlavor( dragRef,
                  (ItemReference)fWindow,
                  (FlavorType) 'PICT',
                  (Ptr)*flavorDataHandle,
                  GetHandleSize((Handle)flavorDataHandle ),
                  (FlavorFlags)0 );
```

We locked the `PicHandle` so we could use its master pointer. Once we return from `AddDragItemFlavor()`, we unlock the handle.

```
  HUnlock( flavorDataHandle );
```

If we encounter a problem, we'll dispose of the drag and return.

```
   if ( err != noErr )
  {
    DisposeDrag( dragRef );
    return;
  }
```

Next, we'll calculate the rectangle we want dragged out of the window. This rectangle has nothing to do with the data we are sending, but just defines the xor rectangle that represents the drag. Try modifying the drag so the dragged rectangle represents the PICT's frame instead. Next, change the drag so it drags the true outline of the picture around instead of just its framing rectangle. Hint: check out the routine `BitMapToRegion()`.

```
  itemBounds = (**((WindowPeek)fWindow)->contRgn).rgnBBox;

  err = SetDragItemBounds( dragRef, (ItemReference)fWindow,
                                               &itemBounds );
  if ( err != noErr )
  {
    DisposeDrag( dragRef );
    return;
  }
```

This code takes a solid region and turns it into just the frame of the region. We then pass the rectangular region on to `TrackDrag()` which drags the rectangle around.

```
  dragRegion = NewRgn();
  RectRgn( dragRegion, &itemBounds );
```

Designed for

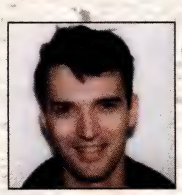
Microsoft®
Windows™95

How ya doin'? They say opportunity knocks. This time opportunity is breaking down the door. Windows™ 95 looks to be a phenomenal success, and your Mac® programming experience has prepared you to be one of the best Windows™ 95 developers in the business.

You love the Mac®, Right? Because early on, it let you develop cool new applications that you couldn't create anywhere else. Applications that delivered a compelling end-user experience. Applications that worked together more closely than ever before. We know; we feel the same way. That's why Microsoft was one of the first and strongest supporters of the Mac, even before it was launched. Last year we shipped more applications for the Mac than anyone else. We still support the Mac—and you should too.

Because you love the Mac experience you should also look at Windows® 95. Intuitive user interface, Plug-n-Play hardware, long file names...they're all there in Windows 95. Throw in pre-emptive multitasking, an integrated object model, and enterprise connectivity...and you've got a very powerful platform for creating a whole new generation of insanely great applications.

You're wondering how to get involved? We're not talkin' about life in the slow lane here. Register for the 10th Annual MacHack today, and learn how your Mac experience will help you join your friends in programming for Windows 95.



We just shipped a Widows 3.1 version of Ray Dream Designer last October and sales have been excellent, rapidly equaling our Mac sales. The market for graphics applications on Windows is wide-open. And because of Widows 95's new development tools and features (such as memory-mapped files, threads, and image color matching), our engineering team is having a lot of fun programming for Windows 95. Come on in; the water's fine!

Eric Hautemont, President, Ray Dream, Inc.

Photo: Andrea Carr-Lopinto

The transition to Windows 95 offers smaller, fast-moving companies the same great opportunities found in the early days of the Macintosh. Windows 95 is where the action is.

Martin Mazner, former Publisher of MacUser Magazine

MacHack

In response to customer demand, MacHack is once again helping its attendees explore their cross-platform development options. This year, MacHack is proud to offer you a two-day, pre-conference seminar on Windows 95 Programming for Macintosh Developers. A two-day version of the one-day seminars you have heard about at MacWorld and the WWDC, the MacHack seminar will go into the depth and detail that the MacHack audience demands. You'll be introduced to Windows 95 by the man who introduced you to the Macintosh—Stephen Chernicoff, author of the seminal *Macintosh Revealed* series. This seminar will show you how to get the most out of:

The Win32 API • Virtual Memory Management
Memory-Mapped Files • Processes and Threads
Structured Exception Handling
Windows 95's User Interface Architecture • OLE
Networking • Development Tools • Code Portability

This MacHack seminar won't turn you into a Windows programming guru overnight, but it will get you off to a running start. And because it's right before MacHack, and in the same location, you can easily attend MacHack as well. The seminar will start on the morning of Tue. June 20, and end the evening of Wed. June 21—right before MacHack kicks off with its keynote address late Wednesday night.

Windows 95 Seminar: Tues., June 20—Wed., June 21
$199 if you pre-register before April 16; $249 thereafter
MacHack '95: Thurs., June 22—Sat., June 24
$375 if you pre-register before April 16; $475 thereafter
Location: Ramada Inn, Southfield, Michigan (fly into Detroit)

To register or for more details, call Expotech at (313) 882-1824, or send email to EXPOTECH on AppleLink or expotech@aol.com on Internet.

Don't WaitNextEvent—HUnlock the Gestalt of Windows 95 today!

```
    tempRgn = NewRgn();
    CopyRgn( dragRegion, tempRgn );
    InsetRgn( tempRgn, 1, 1 );
    DiffRgn( dragRegion, tempRgn, dragRegion );
    DisposeRgn( tempRgn );

    err = TrackDrag( dragRef, eventPtr, dragRegion );
    DisposeRgn( dragRegion );
    DisposeDrag( dragRef );
    return;
}
```

`DragEnterWindow()` gets called when the window receives a `dragEnterWindow` message, indicating that a drag is entering our window. The sender of the drag might be our window, our application, or a different application (like the Finder). We'll first check to see if the incoming drag has PICT data and set `fCanAcceptDrag` accordingly. We'll also set `fIsWindowHighlighted` to false, since we haven't done any highlighting yet.

```
OSErr
TPictureWindow::DragEnterWindow( DragReference dragRef )
{
  fCanAcceptDrag = IsPictFlavorAvailable( dragRef );
  fIsWindowHighlighted = false;

  if ( fCanAcceptDrag )
    return noErr;
  else
    return dragNotAcceptedErr;
}
```

`DragInWindow()` gets called when the window receives a `dragInWindow` message. We'll receive one `dragEnterWindow`, many `dragInWindows`, then one `dragLeaveWindow` messages each time a drag enters our window.

```
OSErr
TPictureWindow::DragInWindow( DragReference dragRef )
{
  DragAttributes  attributes;
  RgnHandle     tempRgn;
```

`GetDragAttributes()` gives us access to the `dragHasLeftSenderWindow` and `dragInsideSenderWindow` flags, which tell us if the drag ever left its originating window and if the drag is currently inside the sending window. Since we don't want a window to receive a drag from itself (from another window in the same application is fine, though) we'll return if the sender is us.

```
  GetDragAttributes( dragRef, &attributes );

  if ( (! fCanAcceptDrag) || (! (attributes &
                         dragHasLeftSenderWindow))
    || (attributes & dragInsideSenderWindow) )
    return dragNotAcceptedErr;
```

If the mouse is in our content region (as opposed to in our title bar – important difference!) we'll highlight the window if it isn't already highlighted.

```
  if ( this->IsMouseInContentRgn( dragRef ) )
  {
    if ( ! fIsWindowHighlighted )
    {
      tempRgn = NewRgn();
      RectRgn( tempRgn, &fWindow->portRect );

      if ( ShowDragHilite( dragRef, tempRgn, true ) == noErr)
        fIsWindowHighlighted = true;

      DisposeRgn(tempRgn);
    }
  }

  return noErr;
}
```

Finally, once the drag focus leaves our window, we'll hide the highlighting and reset `fIsWindowHighlighted` and `fCanAcceptDrag` (just to be safe).

```
OSErr
TPictureWindow::DragLeaveWindow( DragReference dragRef )
{
  if ( fIsWindowHighlighted )
    HideDragHilite( dragRef );

  fIsWindowHighlighted = false;
  fCanAcceptDrag = false;

  return noErr;
}
```

HandleDrop() gets called when our window has accepted a drag and we are now supposed to save the data.

```
OSErr
TPictureWindow::HandleDrop( DragReference dragRef )
{
  OSErr          err;
  Size           dataSize;
  ItemReference  item;
  FlavorFlags    flags;
  DragAttributes attributes;

  GetDragAttributes( dragRef, &attributes );

  if ( attributes & dragInsideSenderWindow )
    return dragNotAcceptedErr;
```

We'll first make sure the first drag item has PICT data. If it does, we'll get the data size, then create a new handle to store the new PICT. Notice that we first try to allocate memory using `TempNewHandle()` which gets its data from outside the application heap. If that fails, we'll go the more normal route of calling `NewHandle()`. This isn't necessarily the right way to do this, since we are stealing memory from other applications, but I thought you might want to play with this call.

```
  err = GetDragItemReferenceNumber( dragRef, 1, &item );
  if ( err == noErr )
    err = GetFlavorFlags( dragRef, item, 'PICT', &flags );

  if ( err == noErr )
  {
    err = GetFlavorDataSize( dragRef, item, 'PICT', &dataSize);
    if  (err == noErr )
    {
      fDraggedPicHandle = (PicHandle)TempNewHandle( dataSize,
                                                    &err );

      if ( fDraggedPicHandle == nil )
        fDraggedPicHandle = (PicHandle)NewHandle( dataSize );
```

If the memory was successfully allocated, we'll load the data

FREE INSTALLER!

To: Mac Developers & Product Managers

From: MindVision (Creators of Stacker for Macintosh)

Subject: Free Copy of DEVELOPER VISE 3.0

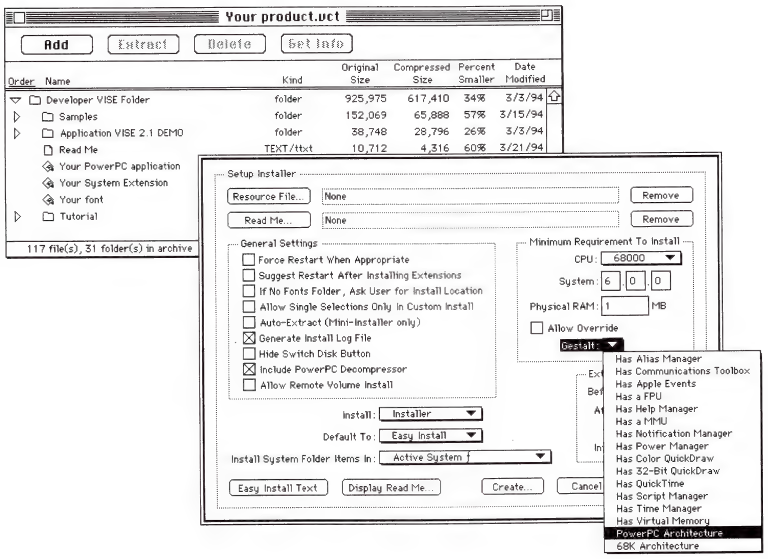
Message: We've got a great new installer for you. Here's your chance to try it for free. No risk, no obligation, no hassle. Call today, don't delay!

Full PowerPC Support

No programming required

Integrated compression

Fully graphical interface



Order Name	Kind	Original Size	Compressed Size	Percent Smaller	Date Modified
Developer VISE Folder	folder	925,975	617,410	34%	3/3/94
Samples	folder	152,069	65,888	57%	3/15/94
Application VISE 2.1 DEMO	folder	38,748	28,796	26%	3/3/94
Read Me	TEXT/ttxt	10,712	4,316	60%	3/21/94
Your PowerPC application					
Your System Extension					
Your font					
Tutorial					

117 file(s), 31 folder(s) in archive

Setup Installer

Resource File...: None

Read Me...: None

General Settings

- ☐ Force Restart When Appropriate
- ☐ Suggest Restart After Installing Extensions
- ☐ If No Fonts Folder, Ask User for Install Location
- ☐ Allow Single Selections Only In Custom Install
- ☐ Auto-Extract (Mini-Installer only)
- ☒ Generate Install Log File
- ☐ Hide Switch Disk Button
- ☒ Include PowerPC Decompressor
- ☐ Allow Remote Volume Install

Minimum Requirement To Install

CPU: 68000

System: 6 . 0 . 0

Physical RAM: 1 MB

☐ Allow Override

Install: Installer

Default To: Easy Install

Install System Folder Items In: Active System f

Call (402) 477-3269

IF YOU PREFER, USE E-MAIL. APPLELINK, AOL: MINDVISION • COMPUSERVE 70253,1437

Join the growing list of companies who use our VISE technology: Adobe, Apple, CE Software, Claris, CompuServe, DeltaPoint, MacroMedia, Radius, Stac Electronics, Symantec, WordPerfect, and many more.

MindVision Software

MindVision Software 840 South 30th St., Suite C, Lincoln, Nebraska 68510
Voice: (402) 477-3269 Fax: (402) 477-1395 AppleLink, AOL: MindVision

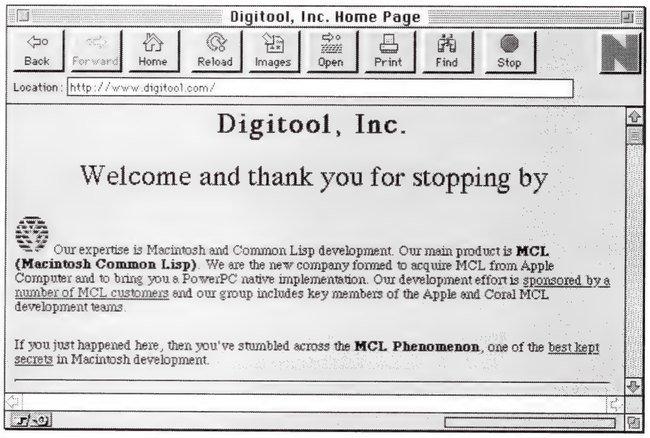
VISE COMPRESSION TECHNOLOGY

Macintosh Common Lisp **Digitool, Inc.**

An Object-oriented Dynamic Language

Take advantage of the
MCL Phenomenon

For the full story visit our Web site:
http://www.digitool.com/



Digitool, Inc.

Welcome and thank you for stopping by

Our expertise is Macintosh and Common Lisp development. Our main product is **MCL (Macintosh Common Lisp)**. We are the new company formed to acquire MCL from Apple Computer and to bring you a PowerPC native implementation. Our development effort is sponsored by a number of MCL customers and our group includes key members of the Apple and Coral MCL development teams.

If you just happened here, then you've stumbled across the **MCL Phenomenon**, one of the best kept secrets in Macintosh development.

PowerPC version forthcoming • Details at Web site

Digitool, Inc. • 675 Massachusetts Avenue • Cambridge, MA 02139
tel: (617) 441-5000 • fax: (617) 576-7680 • email: marketing@digitool.com

from the drag using `GetFlavorData()` then force a redraw assuming we got the data OK.

```
        if ( fDraggedPicHandle == nil )
          err = dragNotAcceptedErr;
        else
        {
          HLock( (Handle)fDraggedPicHandle );
          err = GetFlavorData( dragRef, item, 'PICT',
              *fDraggedPicHandle, &dataSize, 0L );
          HUnlock( (Handle)fDraggedPicHandle );

          if ( err != noErr)
          {
            err = dragNotAcceptedErr;
            DisposeHandle( (Handle)fDraggedPicHandle );
            fDraggedPicHandle = nil;
          }
          else
          {
            SetPort( fWindow );
            InvalRect( &(fWindow->portRect) );
          }
        }
      }
    }

  return( err );
}
```

`LoadDefaultPicture()` loads the PICT resource and drops into the debugger if the PICT resource is missing.

```
PicHandle
TPictureWindow::LoadDefaultPicture()
{
  PicHandle  pic;

  pic = GetPicture( kDefaultPICTResID );

  if ( pic == nil )
  {
    DebugStr( (StringPtr) "\pCould not load PICT resource!" );
    return (PicHandle)nil;
  }
  else
    return( pic );
}
```

`CenterPict()` takes a `PicHandle` and a `Rect` as parameters. On input, the `Rect` contains the window's `portRect`, that is, the rectangle the picture should be centered in. On output, the `Rect` contains a rectangle the size of the picture, but centered in the window and in the window's local coordinate system..

```
void
TPictureWindow::CenterPict( PicHandle picture, Rect *destRectPtr )
{
  Rect windRect, pictRect;

  windRect = *destRectPtr;
  pictRect = (**( picture )).picFrame;
  OffsetRect( &pictRect, windRect.left - pictRect.left,
                windRect.top  - pictRect.top);
  OffsetRect( &pictRect,(windRect.right - pictRect.right)/2,
              (windRect.bottom - pictRect.bottom)/2);
  *destRectPtr = pictRect;
}
```

`IsPictFlavorAvailable()` returns true if the first item in the specified drag contains a PICT flavor.

```
Boolean
TPictureWindow::IsPictFlavorAvailable( DragReference dragRef
)
{
  unsigned short  numItems;
  FlavorFlags     flags;
  OSErr           err;
  ItemReference   item;
```

We'll call `CountDragItems()` to find out how many items are in the drag.

```
  CountDragItems( dragRef, &numItems );

  if ( numItems < 1 )
    return( false );
```

We call `GetDragItemReferenceNumber()` to retrieve the item reference number of the first item. We'll pass that item reference number to `GetFlavorFlags()` to see if the item has a PICT flavor.

```
  err = GetDragItemReferenceNumber( dragRef, 1, &item );
  if ( err == noErr )
    err = GetFlavorFlags( dragRef, item, 'PICT', &flags );

  return( err == noErr );
}
```

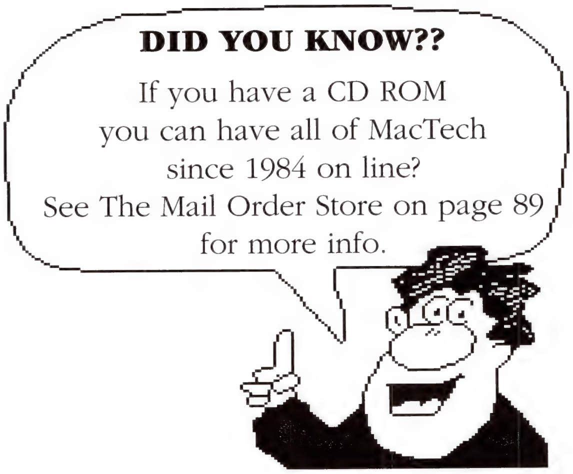
`IsMouseInContentRgn()` returns true if the mouse is in the content region of the window. This is important because we don't want to react to a drag into the title bar of our window. We only react to drags actually inside the window's content region.

```
Boolean
TPictureWindow::IsMouseInContentRgn( DragReference dragRef )
{
  Point  globalMouse;
  OSErr  err;

  err = GetDragMouse( dragRef, &globalMouse, 0L );

  if ( err == noErr )
    return( PtInRgn( globalMouse, ((WindowPeek)fWindow)->contRgn
                                                              ) );
  else
    return( false );
}
```

Till Next Month

I really wish I could have had about twice as much space in this month's column. There is a lot more to the Drag Manager than what I was able to get into here, but this should get you started. The Drag Manager is extremely important. You should definitely support it in all of your applications. A universally available drag and drop capability will help distinguish future Macintosh applications and environments from Windows 95. Though Apple has recently introduced a number of important new technologies (OSA being a prime example), there are none that affect the user's experience as directly as the Drag Manager. In the near future, applications that support the Drag Manager will feel "normal" and those that don't will feel old-fashioned and clunky. The user will expect Drag and Drop support and will likely be pretty unforgiving if it is absent.

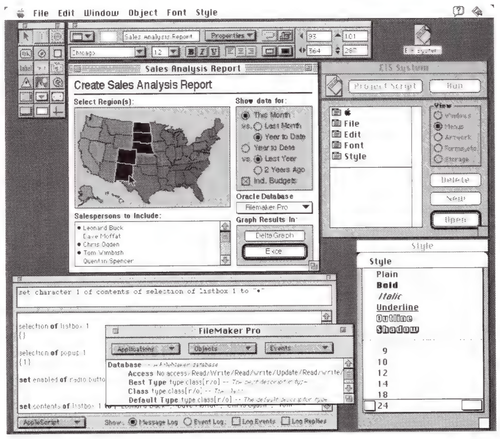
OK, stepping off my soapbox now. Next month, we'll take a look at Sprocket's menu handling model. As I said at the end of last month's column, Sprocket handles menus in much the same way as OpenDoc, so if you learn how to handle menus in Sprocket, you'll have a leg up when you start writing your first OpenDoc part.


DID YOU KNOW??
If you have a CD ROM you can have all of MacTech since 1984 on line? See The Mail Order Store on page 89 for more info.

FACESPAN

The Interface Designer and Application Builder

FaceSpan™ is an extensible Rapid Application Designer (RAD) whose interactive, visual interface design environment and object-oriented AppleScript™ programming assists you in building applications quickly and easily. By supporting AppleScript, FaceSpan allows you to integrate functions from any scriptable program(s) into one application with an interface and functionality customized to your needs.



➤ Develop integrated software from other scriptable apps.

➤ Make stand-alone applications.

➤ Create friendly interfaces for simplified "command centers."

➤ Develop quick prototypes at any level of detail.

Your FaceSpan applications can include any number of windows, dialogs, palettes, and menus. In them, you can display scrolling lists, popup menus, scrolling text, movies, multi-column tables, pictures, icons, buttons, and others. Every interface element has a standard look and feel, but you can script every element to create your own look and feel.

Try what MIS professionals, power users, consultants, and programmers in the know use!

FaceSpan Suggested Retail: $199
Including unlimited runtime user licenses and a FREE UPGRADE to next version for registered users!

SOFTWARE DESIGNS UNLIMITED

Software Designs Unlimited Inc.
1829 East Franklin Street, Suite 1020
Chapel Hill, NC 27514-5861
voice: 919/968-4562 fax: 968-4576
email: FaceSpan@SDU.com
ORDERS: 800-FACESPAN

SYMANTEC TOP 10

SYMANTEC.™

By Steve Howard and Glenn Austin, Symantec Technical Support

This monthly column, written by Symantec's Technical Support Engineers, aims to provide you with technical information based on the use of Symantec products.

Due to the frequency of calls and mail on the following problem, we are repeating a question published earlier.

Q: Why do derived classes of *ifstream* cause a bus error when they read in a stream?

A: Any class indirectly derived from the virtual base class *ios* needs to explicitly call the constructor for *ios* to initialize the data stream's buffer. For example:

```
class myifstream : public ifstream  {
  myifstream(char * s) : ios(&buffer),
             ifstream(s){}
 }
```

Q: When I set project options for new projects, the options do not transfer to any of new projects I create. How do I get the options to remain set when I create new projects?

A: The "new project" option will only work if you remove the project model folder for new projects contained within the (Project Models) folder. The THINK Project Manager will then honor the "new project" options.

Q: I am trying to use Gestalt to find out whether the system has asynchronous SCSI abilities, or whether Quickdraw GX is present, but can't find the right selectors to use. Where do I get this information?

A: There are some Gestalt selectors that are not in GestaltEqu.h. Lists of most known Gestalt selectors and selector codes are available at various ftp sites (see the URL page for possible sites). Once you find the selectors you need, you can add them to GestaltEqu.h, and then re-precompile your MacHeaders, MacHeaders++, and TCLHeaders with the newly-modified GestaltEqu.h.

A simpler way, if your project allows it, would be to add your new selectors into one of your own header files. This would allow you to not need to rebuild your prebuilt header files, as well as allow you to safely replace your GestaltEqu.h with a new one sometime later (without worrying about losing any of your custom-added equates).

To find the answer to your question, we went to MIDnet's Info-Mac searcher (http://www.mid.net/INFO-MAC) and did a keyword search on *gestalt*. We found the following file: ftp://ftp.hawaii.edu/mirrors/info-mac/dev/info/gestalt-selectors-27.hqx and have excerpted a few useful lines (in Pascal terminology):

```
const
     gestaltGraphicsVersion  = 'grfx';  {Gestalt version selector}
     gestaltSCSI             = 'scsi';  {SCSI Manager attributes}
     gestaltAsyncSCSI        = 0;       {Supports asynchronous SCSI}
```

The file we found has much more interesting data, so be sure to go see what other selectors might better suit your purposes.

Q: I want to use the 68881 option in my C++ project. What libraries will I need to recompile?

A: To use the 68881 option, turn on "Generate 68881" for both THINK C and C++, then rebuild the following libraries: CPlusLib, ANSI++, unix++, and IOStreams. Some libraries contain both .c and .cp files, and that's why you'll need to turn on the option for both C and C++.

Q: How do I get my CIconButtons to print?

A: In VA, create a new subclass of CIconButton, called CPIconButton. Change your icon buttons to type CPIconButton, then save and generate. In TPM, open CPIconButton.h, and add the following method to the class:

```
virtual void DrawIcon(Boolean fHilite)
{
    short   drawState = CalcDrawState(fHilite);

 // Set up the correct icon
    if (printing)
        cicnH = nil;
    else
        cicnH = itsCicnH[drawState];

    BlockMove(&itsIcon[drawState], &icon, kIconBytes);
    Prepare();

 // Let base class hilight only if color hiligting
   CIconPane::DrawIcon(fHilite && colorHilite);
}
```

This will force the icon to draw in black & white on the printer, and it will draw correctly on the screen.

Q: I have created a subview that has a CDialogText item and I need a pointer to the item. Can you show me the easiest way to do this?

A: This is a simple two step process. First, get the ID of your CDialogText resource; these IDs are in the generated header file. The header file's name is the subview name with the word "Items.h" appended (e.g. "SubviewItems.h"). Second, call the FindViewByID method, passing this ID.

```
#include "SubviewItems.h"   // list of resource IDs for objects in the subview

CDialogText *myText;
// kMyDialogTextID is found in SubviewItems.h
myText = (CDialogText*)fMain_SubviewName->
                              FindViewByID(kMyDialogTextID);
```

Q: I am trying to convert a Pascal record that has a data type of `Byte` into a C++ structure. I am using the C type `char` as the equivalent of `Byte`. Why am I getting bizarre results?

A: The Pascal type `Byte` is equivalent to an `unsigned char` in C or C++.

Q: How do I get CTable to show only 3 columns, no matter what the size of the window?

A: To make the table show a constant number of columns, override the Draw method and resize the columns based on the current width of the window. For example:

```
void CDerivedArray::Draw(Rect *area)
{
        LongRect r;

        GetFrame(&r); // Get current frame of table pane

        static int oldWidth=0;

        int width=r.right-r.left;  //find width

        if(width!=oldWidth)   //If width hasn't changed, don't do anything
            {
                int numCol=GetColCount();
                for(int x=0;x<numCol;x++)
                   {
                        SetColWidth(x,width/numCol);
                   }

                oldWidth=width;  //Update width
            }

         inherited::Draw(area);   //Draw table
}
```

Q: I am using the default console window and want to change the foreground and background colors. Is there any way that THINK C will allow me to do that?

A: You cannot change the foreground or background colors on the default console window. You can to control some aspects of the console like the text font, size and face. You can also control how input to the console window is read. For more information on using the console window, you can refer to the Online Documentation that came with your THINK C/ Symantec C++ product or in THINK Reference under "Console Package Intro" and "Console Input Mode".

Q: I have created a Visual Architect project, added some views and selected generate all. I have not added any of my own code but, when I try to bring the project up to date, I get an error -192 (resource not found). If I have not added code, what causes this error?

A: In VA, you will get this error if you have defined commands that have the following format in the Command Dialog:

```
In Class:  CMain
Do:        Open
View:      None
```

When that command is generated, VA will attempt to create an Open command and associate it with an existing `'CVue'` resource. Since you have not assigned a view to the command, there is no corresponding `'CVue'` resource. Assign a view to the command to alleviate the problem.

Special thanks to: Kevin Quah, Chris Prinos, Scott Morison, Yuen Li, Kevin Irlen, Michael Hopkins, Rick Hartmann, Colen Garoutte-Carson, Craig Conner, Celso Barriga, and Mark Baldwin

MacForth Plus & Power MacForth

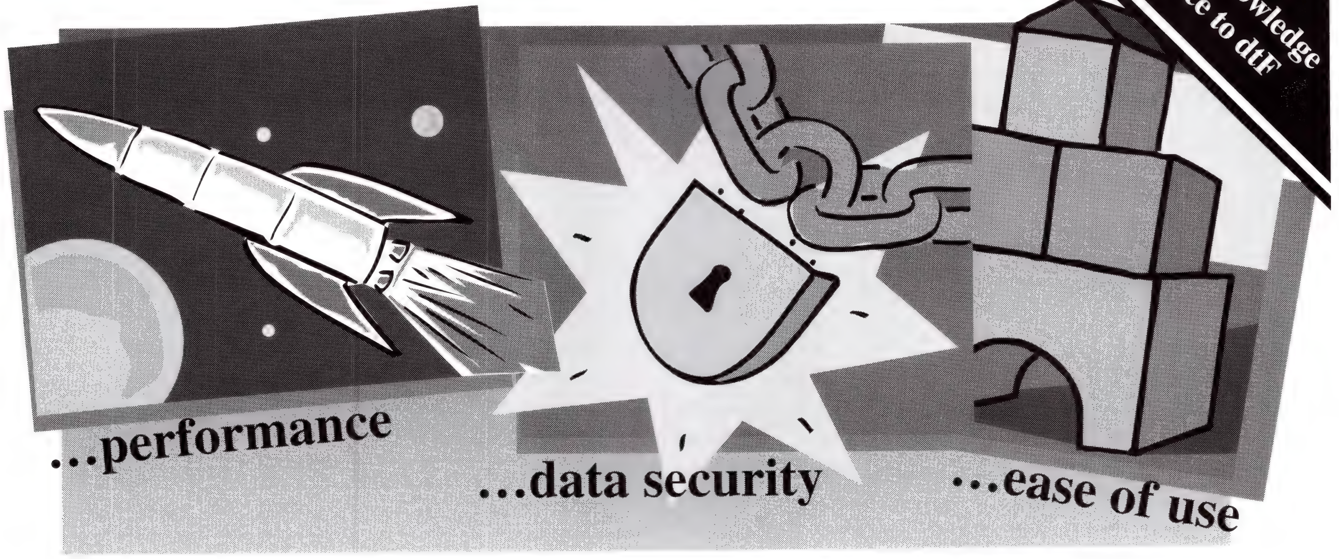
The Language of Innovation is now available native on the Power Macintosh.

FEATURES
- Royalty Free Turnkey Compiler
- Text Editor
- Online Glossary Tool
- System 7 Compatibility
- 68020 Assembler
- 68881/2 Co-processor Support
- High Level Graphics
- Toolbox Support
- Extensive Documentation
- Reasonable Upgrades
- Optional Tools Disks
- MacForth Plus $199
- Power MacForth $199

Creative Solutions, Inc.
4701 Randolph Road, Suite 12
Rockville, MD 20852
301-984-0262 or 1-800-FORTH-OK *(orders)*
Fax: 301-770-1675 Applelink: CSI

dtF The Relational Database System

New!
DataScript™ by General Knowledge
The AppleScript-Interface to dtF



When it comes to performance **dtF** is in a class all its own. **dtF** utilizes a proprietary query optimization and caching scheme to obtain unparalleled performance. And if you are in need of a quantum leap in performance, use the **dtF** native PowerPC standalone or server version.

Full transaction control and error recovery guarantee maximum data protection even after sudden system crashes. **dtF** databases are compressed and encrypted to protect against all unauthorized access, even disk editors.

Integrated data dictionary, security, automatic index selection, query optimization, deadlock detection and error recovery allow you to concentrate on your application. The **dtF** API was designed for a seamless integration with third-party toolkits and class libraries.

dtF features include SQL, BLOB support, electronic documentation, sample applications, tutorials, and tools for importing, exporting, creating, managing databases and users. **dtF** is currently available for Macintosh, DOS, Windows, OS/2. The C/C++ API is identical across all platforms.

Available for Macintosh System 7.x and native PowerPC. MPW C/C++ (68K and native PPC), Symantec C/C++, Metrowerks CodeWarrior (68K and native PPC). Separate versions for use with HyperCard, SuperCard, SmalltalkAgents® and Prograph CPX available.

Standalone applications built with dtF are royalty-free.

dtF Americas, Inc.
19672 Stevens Creek Blvd.,
Suite 128
Cupertino, CA 95014, USA

Phone:	(800) DTF-1790
Fax:	(510) 828-8755
AppleLink:	DTF.AMERICA

dtF The Relational Database System

INTERNET SYSTEM SOFTWARE

By Quinn "The Eskimo!", <internet-config@share.com>

Using the Internet Configuration System

Use IC to access common Internet-related user settings.

How many times have you entered your Email address into different Internet-related applications? What about the file type and creator of your preferred viewer for JPEG files? If you're a dedicated net.junkie like I am, you'll find yourself using dozens of Internet-related applications, each with its own set of preferences tucked away in a multitude of obscure dialogs.

This situation gets even worse when you want to change this information. I'm sure you've had the pleasure of trying to make sure that your mail and news signatures are always synchronised, but that is easy in comparison to changing your preferred JPEG viewer in Fetch, MacWeb, StuffIt Expander, uuUndo, and so on.

Now multiply this wasted time by the millions of Macintosh users hooked up to the net today, or who will hook up in the next few years. That's an awful lot of redundant information being rekeyed.

As a programmer, I'm sure you'll realise that it is not just the users who are wasting time and energy here. While it is relatively easy to save, restore, and provide the user interface to change simple preferences like an Email address, the programming effort required to implement a file-extension-to-type mapping dialog is considerable. Which leads to the ridiculous situation where many small Internet programs do not provide any user interface for this; instead they use a hard-wired table or offer configuration only through ResEdit.

"Oh, by the way, the full source code to all parts of the Internet Config System is in the public domain."

The Internet Configuration System is designed to solve these problems by providing a common database for user settings. Users employ the Internet Config application to set their preferences in this database and you can use a standard Application Programmer Interface (API) to get these preferences for use in your code.

This article describes how to use the Internet Configuration System, or IC for short, to access Internet-related user settings. It starts with a brief history of IC and then jumps straight into code, starting with the simplest possible IC aware application and then moving on to a useful sample program. Next comes a checklist for making your program IC aware and a discussion of some of the gotchas of that process. Then it takes a brief look at IC's internals and concludes with a glimpse into the future of IC.

Quinn – Quinn works as a programmer for the Department of Computer Science at The University of Western Australia. Programmers perform a variety of tasks, which occasionally include programming. When not programming at work, Quinn writes programs for fun, and then gives them away. Only other programmers understand that last bit. Quinn has co-authored a number of programs with Peter N Lewis, including the Internet Configuration System.

neoAccess™

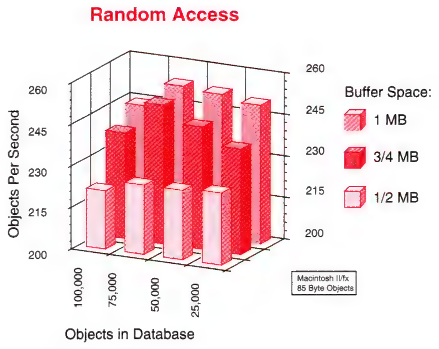
Cross-Platform Object Database Engine

Full Featured

NeoAccess™ allows you to embed the power of a full-featured object-oriented database engine into your **Macintosh**, **Windows** and **DOS** based applications. No other object database engine can match NeoAccess's impressive feature set, including: Blobs, part lists, iterators, swizzlers, temporary objects, multiple indices on a class, schema evolution, a powerful relational query mechanism, a streams-based i/o model and incredible performance.

High Performance

Internally, NeoAccess uses extended binary trees and binary search algorithms to achieve optimally short access times. Its automatic query optimizer ensures that queries always use the fastest access path to objects. And indices are dynamically combined, collapsed and compressed to keep access times to an absolute minimum as the contents of a database changes. NeoAccess's object caching boosts performance by keeping objects in memory even after being disposed of by the application. Your application's memory size is reduced because only those objects of immediate interest need to be in memory at any one time, not the entire file.



Full Source Code

We've taken a frameworks approach toward object persistence and database technology. In much the same way that application frameworks are used to construct the front-end of an application, NeoAccess is the framework you use to build your application's back-end. As is the case with virtually all object frameworks, the NeoAccess Developer's Toolkit comes complete with ***full source code***, for all major application frameworks including Metrowerks' **PowerPlant**, Symantec's **THINK Class Library** and Apple's **MacApp** on the Macintosh and **Microsoft's Foundation Classes**, Inmark's **zApp** and Borland's **ObjectWindows** in Intel-based environments. It can even be used without a framework or in one that you've designed.

Easy to Use

The programming interface is designed around the concept of minimum visible complexity. Application-specific objects inherit persistence properties from a NeoAccess base class. These objects are organized in the database primarily by class. But NeoAccess also knows how classes are related. So multiple classes can be searched in a single operation. And of course objects of any particular class can be organized using multiple indices. NeoAccess is unique in that it allows objects to be located based on abstract selection criteria or based on their relationship to other objects. There's literally no database administation to deal with – NeoAccess takes care of all the details. NeoAccess also includes a Blob mechanism which allows free-form variable-length data to be stored in databases with the same ease as fixed-length objects. NeoAccess even includes a powerful set of keyed iterators for traversing indices and part lists. Keyed iterators have the unique ability to iterate over only those objects in a set that match a given selection criteria. Your users will appreciate NeoAccess because databases are completely self-contained in a single document file. So users can treat a database file as they would any other document.

Proven

NeoAccess has been commercially available since May of 1992. Hundreds of commercial and in-house applications based on NeoAccess technology have already been deployed. NeoAccess can help your organization deliver powerful products in a more timely fashion than you ever imagined possible.

Affordable

NeoAccess's best feature is its price. The NeoAccess Developer's Toolkit sells for just $749 per developer with absolutely ***no runtime licensing fees***. It includes ***full source code***, numerous sample applications, 450+ pages of documentation, and 30 days of technical support. So what are you waiting for?

Now on the CodeWarrior CD!
See the NeoLogic folder on the root of the CD-ROM for more details.

neo•logic
In order to survive, you need to persist.

No Runtime Licensing Fees!
Use NeoAccess in all your development for one low price!

1450 Fourth St. • Suite 12 • Berkeley • CA • 94710 • (510) 524-5897 • neologic@mail.dnai.com

A Brief History of Internet Config

The Internet Config project has its roots in March 1994 when a discussion started in the USENET newsgroup comp.sys.mac.comm. The idea was proposed and the general consensus was "Hey, that's a very good idea, we should do it." Obviously they had no idea how much work it would be! Peter N Lewis moved the discussion to mail by creating the Internet Config mailing list, and added everyone involved in the discussion to the list without even asking them. Fortunately we didn't get too many complaints about the unilateral action, despite the fact that some of the US correspondents arrived to work that morning with over 40 IC-related messages!

"Internet Config Mailing List"

The Internet Config mailing list is dedicated to the discussion of the technical details of the Internet Configuration System. You can subscribe to the list by sending mail to <listserv@list.peter.com.au> with the body of the message containing "subscribe config Your Real Name".

A fortnight of in-depth discussion followed, culminating in the design and implementation of the first IC API. Later that month we distributed the first implementation of the Internet Config component, the core of which has remained fairly stable since that time.

The project then went into limbo, partly because the movers and shakers all disappeared to WWDC '94, but mainly because the big job in front of us was writing the Internet Config application, which was mostly user interface code that no one wanted to tackle. We procrastinated for almost six months before eventually resuming the project in September 1994. After burning months of Friday nights on the project (social life, what's that?), we shipped the first beta of the Internet Config System proper on 30 October 1994. After going through the usual sequence of betas, we shipped Internet Config 1.0 to the world on 4 December 1994.

The Simplest IC Program

It is traditional to implement a "Hello World!" program in whatever new programming language or tool invented. Well maybe Western Australians are more cynical than others but around here the tradition is to have the program say "Hello Cruel World!". This section shows how to do just that with IC and Think Pascal. In the process it demonstrates the Internet Config user experience.

Before beginning, I should point out that, although the examples here are in Think Pascal, Internet Config can be called from all common Macintosh development environments including: Think Pascal and C; MPW Pascal and C; and Metrowerks Pascal and C for both 68K and PPC.

To start with, run the Internet Config application, which displays the window shown in Figure 1.

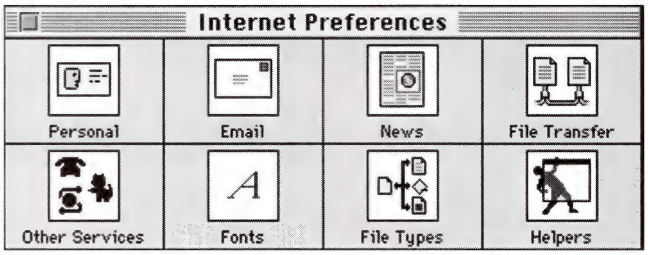
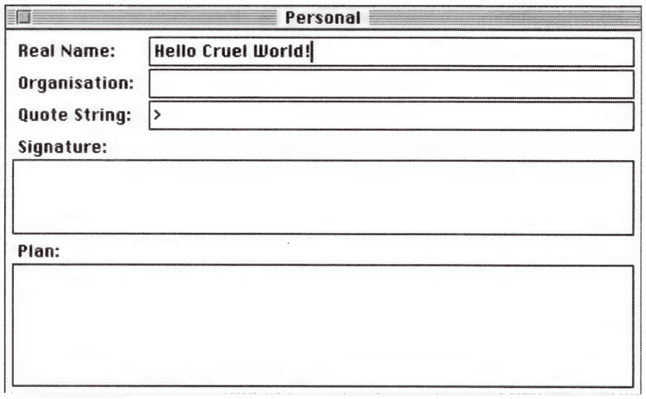


Figure 1. The Internet Config application main window

If this is the first time you have run IC, it will ask you if you want to install the Internet Config Extension. You should agree to this even though (due to the magic of IC) the example will still work if you don't.

Each of the buttons on the main window opens another window displaying a group of preferences. For this exercise, click the Personal button to open the window shown in Figure 2, and then enter "Hello Cruel World!" into the Real Name field.



Real Name: Hello Cruel World!
Organisation:
Quote String: >
Signature:
Plan:

Figure 2. The Internet Config Personal preferences window

A user would normally go through and set up the rest of their preferences but for now just close this window and save your changes. By default IC stores your preferences in a file called "Internet Preferences" in the Preferences folder.

You are now ready to start coding!

Listing 1: ICHelloCruelWorld.p

ICHelloCruelWorld

```
program ICHelloCruelWorld;
  (* ICHelloCruelWorld *)
  (* The simplest IC aware program.  It simply outputs *)
  (* Real Name preference, which is assumed to contain *)
  (* the text "Hello Cruel World!" *)

  uses
    ICTypes, ICAPI, ICKeys;   (* standard IC units *)

  var
    instance: ICInstance;
              (* opaque reference to IC session *)
    str: Str255;
```

```
                    (* buffer to read real name into *)
      str_size: longint;
                    (* size of above buffer *)
      junk: ICError;
                    (* place to throw away error results *)
      junk_attr: ICAttr;
                    (* place to throw away attributes *)
begin
   (* start IC *)
   if ICStart(instance, '????') = noErr then begin
      (* specify a database, in this case the default one *)
      if ICFindConfigFile(instance, 0, nil) = noErr then begin
         (* read the real name preferences *)
         str_size := sizeof(str);
         if ICGetPref(instance, kICRealName, junk_attr,
                   @str, str_size) = noErr then begin
            writeln(str);
         end; (* if *)
         (* shut down IC *)
         junk := ICStop(instance);
      end; (* if *)
   end; (* if *)
end. (* ICHelloCruelWorld *)
```

There are four important lines in this program, each of which corresponds to a critical API call.

1 ICStart

Before using IC, your program must call ICStart, which initialises the system and returns an instance, which is your program's connection to the system. Your program passes this instance back to all other API calls. The instance is a completely opaque type; all you know about instances is that IC hangs its internal state off them.

The term instance is inherited from IC's internal design, which uses the Component Manager to implement a simple form of dynamic linking. See the later section, "Under the Bonnet", for more details.

2 ICStop

Each successful call to ICStart must be balanced with a call to ICStop. IC uses this call to clean up after itself, disposing any memory that it has allocated and so on. ICStop effectively disposes the instance so be careful not to use it afterwards.

3 ICFindConfigFile

This call instructs IC on how to find the appropriate preference information. You can use this call to specify a preference search path but this example uses the default search path and so it gives simple default values for both the parameters.

4 ICGetPref

This call actually reads the preference data into a buffer that you supply. At this stage it is important that you understand something about preferences. Each preference has three conceptually unique parts. The first part is the **key**. A key is a plain text Str255 that uniquely identifies the preference. The key for the real name preference is "RealName", although this example uses the constant kICRealName.

The second part of each preference is the **value**. The value of a preference is an arbitrarily long untyped block of data. The interpretation of this data is up to the application, although most of the existing preferences use common data structures, such as a Pascal string, for preference values.

The third part of each preference is the **attributes**. Each preference has one long word of attribute bits associated with. These attributes are used to store supplementary information about the preference. It is easy to write your program so that you can completely ignore attributes.

The parameters to the ICGetPref call in Listing 1 should now be obvious. The instance is the connection to IC that is common to all API calls. The kICRealName is a Pascal string that identifies the preference required. The junk_attr is a placeholder for the preference's attributes which are returned and subsequently ignored. The pointer parameter is the address of the buffer where the preference value should be copied and the str_size parameter is the size of that buffer.

The Internet Config API contains eleven other routines, but all of the important functionality is embodied in the four routines demonstrated here. You can find out more about the API by reading the Internet Config Programming Documentation.

The Internet Config Programmer's Kit is included on the disk that accompanies this magazine. The latest version of the kit is kept on the following FTP sites:

```
ftp://ftp.share.com/internet-configuration/
ftp://redback.cs.uwa.edu.au//Others/Quinn/Config/
```

A Useful Sample

"Hello Cruel World" programs are, by their very nature, contrived examples. However the program in Listing 1 is actually a good demonstration of the level of technology required to use IC in a Real World™ situation. Accessing simple preferences really is that simple.

A more complicated situation arises when you want to work with the Mappings preference. This preference contains the information required to map file name extensions (such as .jpg) to their file type and creator (such as JPEG/JVWR). The value of this preference is a complicated table of entries, with each entry giving one mapping.

Fortunately, the IC system comes bundled with a library that lets you parse this table easily. The program shown in Listing 2 demonstrates this technique.

Listing 2: SpaceAliens.p

```
program SpaceAliens;
   (* Space Aliens Ate My Icons *)
   (* A drag and drop utility to fix the type and *)
   (* creator of any dropped on file based on its *)
   (* extension and the database of extension mappings *)
   (* provided by Internet Config. *)
```

The database engine for professionals

CXBase Pro

Based on the database engine used by **Apple AOS** for their **eWorld** content publishing software, *CXBase Pro* represents the state of the art in data storage and retrieval technology. *CXBase Pro* has all the tools you need to develop any kind of application that needs its own database engine.

The philosophy behind *CXBase Pro* is to provide a simple, consistent, and powerful API that lets you work in whatever way you choose. *CXBase Pro* is delivered with full source code, giving you maximum flexibility and security. The *CXBase Pro* library provides both multi-keyed data record handling, and flexible BLOB storage where you define the data format. And *CXBase Pro* has been engineered from the ground up for maximum performance.

Powerful and flexible select module

CXBase Pro lets you create multiple kinds of selections, and provides a callback mechanism that allows you to both display the progress, and capture the data as it is selected.

Native PowerPC version

What better way to use the speed of a Power Macintosh than to rev up your data access? Average native speed 3 times that of the fastest Quadra.

100% Cross-platform

Both the source and data files are compatible across all platforms. Yes, Virginia, it really works!

Access to logical and absolute record numbers

You can access data records not only by key, but also by logical and absolute record numbers.

Royalty–free license

You pay once for the development license. Period!

Test drive before you buy - Full-featured $49 demo available!

CXBase Pro also comes in a demo version. You get the complete manual, and a full-featured library limited only by file size. You get access to all the function calls, and you can use it as long as you like. If you decide to buy the full package you get credit for the price of the demo.

For full prices, to order, or to receive more information, contact:

TSE International | Taandwarsstr. 51 | 1013 BV Amsterdam, Holland | tel: 31 20 638-6507 | fax: 31 20 620-4933 | AppleLink :TSE.INT

Apple, AOS, eWorld, Power Macintosh, and Quadra are trademarks of Apple Computer, Inc.

```
uses
  (* standard system units needed to do AppleEvents *)
  (* remember that Think Pascal automatically uses *)
  (* most of the base operating system *)
  EPPC, AppleEvents,

  (* standard IC units *)
  ICTypes, ICAPI, ICKeys,

  (* bonus IC units, extra libraries shipped as source code *)
  ICMappings, ICSubs;
```

GotRequireParams

```
function GotRequiredParams (theAppleEvent: AppleEvent): OSErr;
  (* standard AppleEvent routine copied out of NIM:IAC *)
  var
    typeCode: DescType;
    actualSize: Size;
    err: OSErr;
begin
  err := AEGetAttributePtr(theAppleEvent,
             keyMissedKeywordAttr, typeWildCard,
             typeCode, nil, 0, actualSize);
  if err = errAEDescNotFound then begin
    GotRequiredParams := noErr;
  end
  else if err = noErr then begin
    GotRequiredParams := errAEEventNotHandled;
  end
  else begin
    GotRequiredParams := err;
  end; (* if *)
end; (* GotRequiredParams *)
```

Global Declarations

```
const
  my_creator = 'SA8I';
        (* the application signature *)
var
  quit_now: boolean;
        (* set to true when you want main loop to quit *)
  instance: ICInstance;
        (* global connection to IC *)
  mappings: Handle;
        (* the mapping preference as returned by IC *)
```

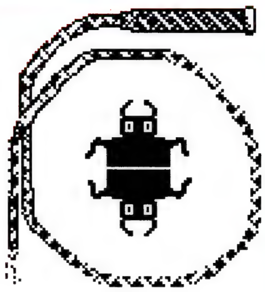
ProcessDocument

```
function ProcessDocument (fss: FSSpec): OSErr;
  (* this is the core of the program *)
  (* the fss parameter is a file whose extension we'll look up in the IC database *)
  (* mappings global variable is already set up to contain that database *)
  var
    err: OSErr;
    count: longint;
          (* total number of entries in database *)
    i: longint;
          (* indexes over the database entries *)
    this: ICMapEntry;
          (* an unpacked element of the *)
          (* mappings database, used while stepping *)
          (* through database *)
    entry: ICMapEntry;
          (* a mappings database element *)
          (* used to record the best match *)
    longest_len: integer;
          (* longest extension we've found so far *)
    posndx: longint;
          (* the index into the mappings database *)
    info: FInfo;
          (* temporary for changing type and creator *)
```

Gives you the Information to *Program your Best!*

The Debugger V2 & MacNosy

by Steve Jasik



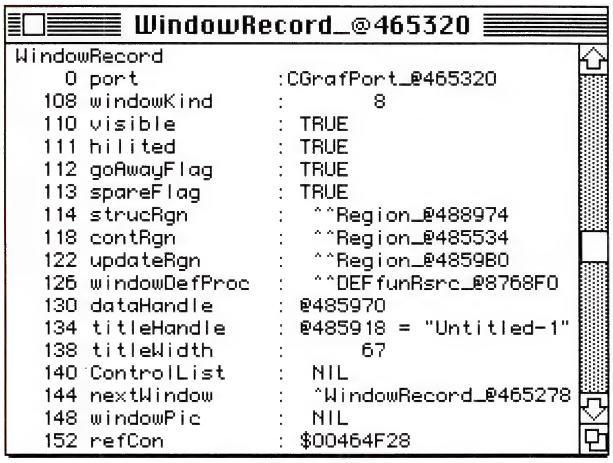
The Debugger is a low and high-level symbolic Debugger that runs in a full multi-window Macintosh environment. You can trace program execution, view the values of variables, etc. ***of both 68K and PowerPC programs***.

MacNosy is a global interactive disassembler that enables one to recover the source code of any Mac application, resource file or the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system (6.0x or System 7.x) environment symbolically!

It is the *only* debugger to use the MMU to protect your CODE resources and the rest of the system from the program you are debugging. With MMU Protection you can find errors when they happen, not millions of instructions later! (Macintoshs with 68030 CPUs only).

The Debugger is the debugger of choice at: Adobe, Aldus, Claris, Electronic Arts, Kodak, Metrowerks, etc.



```
WindowRecord_@465320
WindowRecord
    0 port            :CGrafPort_@465320
  108 windowKind      :       8
  110 visible         :  TRUE
  111 hilited         :  TRUE
  112 goAwayFlag      :  TRUE
  113 spareFlag       :  TRUE
  114 strucRgn        :   ^^Region_@488974
  118 contRgn         :   ^^Region_@485534
  122 updateRgn       :   ^^Region_@4859B0
  126 windowDefProc   :   ^^DEFfunRsrc_@8768F0
  130 dataHandle      :  @485970
  134 titleHandle     :  @485918 = "Untitled-1"
  138 titleWidth      :       67
  140 ControlList     :   NIL
  144 nextWindow      :   ^WindowRecord_@465278
  148 windowPic       :   NIL
  152 refCon          :  $00464F28
```

An example of a structured data display window

Its Features Include:

- **Symbolic Debugging** of *any* Macintosh program, ROM, or *code resource* (DRVRs, XCMDs, INITs, PDEFs, 4DEXs ..)
- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...), and an Incremental Build System with instant Link for superfast development.
- **Object Inspector for MacApp 3 programs**
- **Source level debugging of Think C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Fast Software Watchpoint** command to find clobbered variables
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Handle Zapping, Heap Scramble and Heap Check to detect program errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to help filter out redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, – make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: $350
Runs on all Macs. Call For Group prices or Updates. Visa/MC Accepted.

Available from: Jasik, APDA, Frameworks or ComputerWare (800-326-0092).

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • **(415) 322-1386**
Internet: macnosy@netcom.com • Applelink: D1037

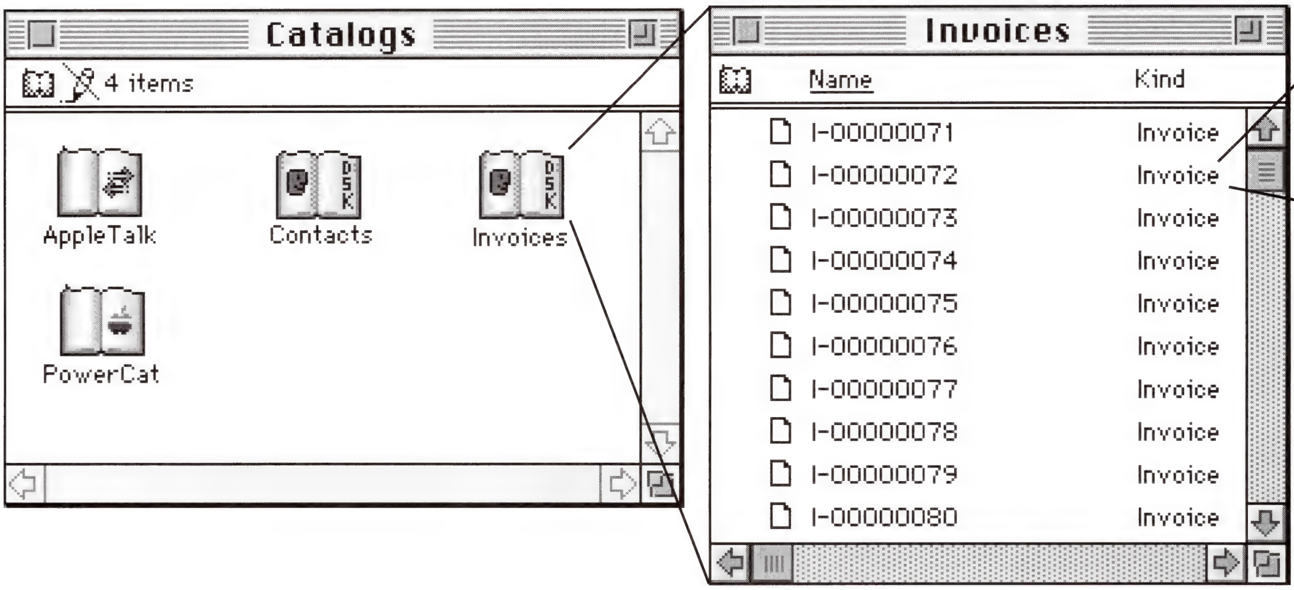
Databases in the Finder?

Unfortunately, most users must interact with relational databases at the SQL level or behind a front-end that can take months to develop.

select invoice_num,invoice_date,cust_id,cust_name from invoices,customers where invoice_id = 112;printall;

Why not view and edit your data at the Finder level? Announcing **Cataloger**™.

Cataloger brings a whole new way of working with databases to the Macintosh. The Finder™ has always provided a view of our desktop files, why not database records too?



Name	Kind
I-00000071	Invoice
I-00000072	Invoice
I-00000073	Invoice
I-00000074	Invoice
I-00000075	Invoice
I-00000076	Invoice
I-00000077	Invoice
I-00000078	Invoice
I-00000079	Invoice
I-00000080	Invoice

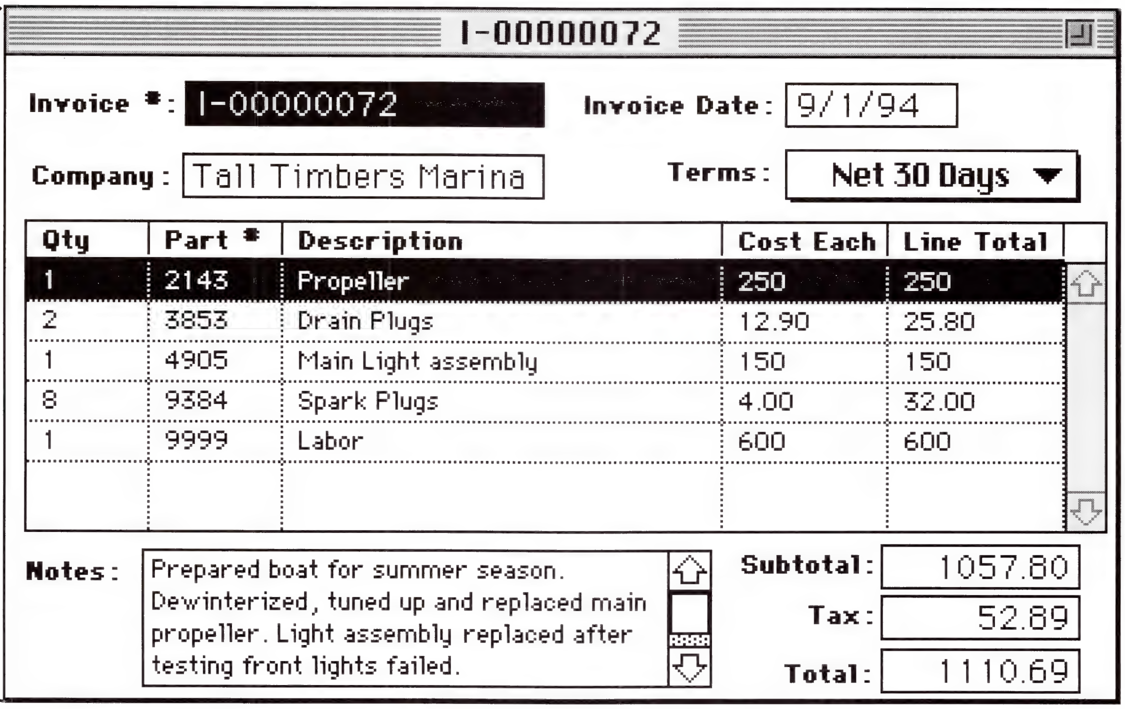
Apple's **O**pen **C**ollaborative **E**nvironment places a catalog icon at the Finder. Open the icon up and you'll see a window like the one above. The AppleTalk catalog shows nodes on the network. The PowerCat catalog shows users and groups in a PowerShare Server.

The Contact and Invoices catalogs (for example) can show information from **4D Server**, **DAL/DAM** , **Oracle** and **Sybase** databases! If your needs are more customized, you can write AppleScript handlers to provide the contents of your catalog.

The Catalog Manager (an AOCE API) allows for **C**atalog **S**ervice **A**ccess **M**odules (CSAMs) to be created that interact with external sources.

The Finder and other applications make requests through the Catalog Manager and Cataloger does the rest. There are no limits to the amount of data brought back.

Templates provide the ability to view and edit individual records. AOCE Templates are designed with **Template Constructor™.** This powerful application let's you easily design forms to view data coming from several sources, control behavior with C and AppleScript, and view row/column data with **TableIt!™** - a powerful data display tool.



I-00000072

Invoice #: I-00000072 **Invoice Date:** 9/1/94

Company: Tall Timbers Marina **Terms:** Net 30 Days

Qty	Part #	Description	Cost Each	Line Total
1	2143	Propeller	250	250
2	3853	Drain Plugs	12.90	25.80
1	4905	Main Light assembly	150	150
8	9384	Spark Plugs	4.00	32.00
1	9999	Labor	600	600

Notes: Prepared boat for summer season. Dewinterized, tuned up and replaced main propeller. Light assembly replaced after testing front lights failed.

Subtotal: 1057.80

Tax: 52.89

Total: 1110.69

Access information and catalog definitions are stored in and protected by the AOCE **Keychain**. Your users can have one username and password for a variety of data sources!

Need more horsepower? **Database Scripting Kit**™ provides AppleScript access to all of the connection information and data sources. Write AppleScript routines that send queries and parse results from within your own application.

Cataloger/Template Constructor includes:

Database Scripting Kit™ with connectors for 4D Server, DAL/DAM, Oracle, Sybase and others.

DSK Cataloger™ CSAM

Template Constructor™ with TableIt!™

Several example catalogs/templates and over 100 example AppleScripts.

Complete scriptability, native for Power Macintosh. Drag Manager and Thread Manager support.

How to contact us:

Graphical Business Interfaces, Inc.

Voice: 800-424-7714 or 219-253-8623

Fax: 219-253-7158

E-Mail: steve@gbi.com

GBI. Bringing your data into view.™

Power Mac and Macintosh Developers:

FIND OUT FAST WHAT'S GOING ON IN MEMORY

THE MEMORY MINE™

- See memory allocation in any open heap at a glance.
- Easily spot memory leaks.
- Flags heap corruption when it happens.
- Works with source level debugger to let you find memory problems fast.
- Stress applications on the fly with Purge, Compact, and Zap.
- Allocate memory at will for precise stress testing.
- Log heap data - easily document heap status over time.
- No need for source code: nothing inserted in code; no patches to the system.
- Works with 24-bit, 32-bit, and modern memory managers.

For Macintoshes with 68020 or better. Requires System 7.0 or later.

only $99 US
Order now from Adianta, Inc.
Phone: (408)354-9569 • FAX: (408)354-4292
AppleLink:ADIANTA • AOL:Adianta • Internet:adianta@aol.com
For VISA, MC, or American Express orders by mail, fax, or Applelink, please include name, address, card number, expiration date, and phone number or email address.
Also available through the MacTech Mail Order Store.

for more information contact
Adianta, Inc. • 2 N. Santa Cruz Ave. #201 • Los Gatos, CA 95030

800 Telephone Services ◆ Advertising Agencies ◆ Bar Coding Equipment &
Systems ◆ Documentation ◆ Bulletin Board Systems Software ◆ CD ROM
Mastering Equipment ◆ Clipping Bureaus ◆ Software Manufacturing ◆
Consultants (42 subcategories) ◆ Content Rights Clearance Agencies
◆ Content Providers – Photo ◆ Development Tools ◆ Content Providers ◆
Content Provi ◆ Degaussing
Equipment ◆ subcategories)
◆ Legal ◆ Di Equipment ◆
Disk/CD ROM s ◆ Personnel
◆ Documentat Companies ◆
Executive Rec pth Interviews
◆ Fulfillment k Software ◆
Localization ◆ ustry Research
Companies ◆ gn Consultants
◆ Interactive Consultants ◆
Marketing & iquidators ◆
Localization S s & Managers
◆ Marketing R Multimedia ◆
OEM Service ne Services ◆
Package Desi ◆ Packaging
Materials & S blic Relations
Services ◆ Re re ◆ Search &
Retrieval Soft Distributors ◆
Software Ma s ◆ Financial
Manufacturing trol & Testing
◆Security ◆ ng Research &
Selling Services ◆ Upgrades Consult ◆ Third Party Co-Marketing Programs
◆ Trade Show Services & Exhibits ◆ Trademark Attorneys ◆ Training &
Education ◆ Translators & Interpreters ◆ Software Development Tools ◆
Technical Support ◆ User Groups ◆ User Interface Consultants ◆ Virtual
Reality Consultants ◆ Employment Agencies ◆ Fax on Demand ◆ and More!

The SoftPub Yellow Pages

Directory of Suppliers to the Software Publishing Industry

800-763-8782

206-852-7440

```
begin
  (* count the total number of entries *)
  err := ICMapErr(ICMCountEntries(mappings, count));
  if err <> noErr then begin
    count := 0;
  end; (* if *)
  (* loop through the entries looking for the longest match *)
  longest_len := 0;
  posndx := 0;
  for i := 1 to count do begin
    (* ICMGetEntry gets the entry from mappings *)
    (* that starts at posndx *)
    (* and puts it into the entry record *)
    if ICMGetEntry(mappings,
                   posndx, this) = noErr then begin
      (* increment posndx so that we get the next *)
      (* entry the next time around the loop *)
      posndx := posndx + this.total_length;
      (* the entry matches if *)
      (* not_incoming flag bit is clear *)
      (* it's longer than the previous max *)
      (* it's longer than the file name *)
      (* it matches the last N chars of the filename *)
      if not btst(this.flags, ICmap_not_incoming_bit)
        & (length(this.extension) > longest_len)
        & (length(this.extension) < length(fss.name))
        & (IUEqualString(copy(fss.name,
                              length(fss.name)
                                - length(this.extension)
                                + 1,
                              255),
                         this.extension) = 0)

      then begin
        (* record the new longest entry *)
        entry := this;
        longest_len := length(this.extension);
      end; (* if *)
    end; (* if *)
  end; (* for *)

  (* if we found any matches then *)
  (* set the file type and creator appropriately *)
  if longest_len > 0 then begin
    err := HGetFInfo(fss.vRefNum, fss.parID,
                     fss.name, info);
    if err = noErr then begin
      info.fdCreator := entry.file_creator;
      info.fdType := entry.file_type;
      err := HSetFInfo(fss.vRefNum, fss.parID,
                       fss.name, info);
    end; (* if *)
  end
  else begin
    err := noErr;
  end; (* if *)

  quit_now := true;
  ProcessDocument := err;
end; (* ProcessDocument *)
```

HandleOpenApplication

```
function HandleOpenApplication (theAppleEvent: AppleEvent;
                     reply: AppleEvent;
                     refcon: longint): OSErr;
  (* the 'oapp' event handler, displays the about box *)
  (* should most probably only do this if we're in *)
  (* the foreground but that's just too complicated *)
  (* for this example *)
  var
    err: OSErr;
    email_address: Str255;
    junk_attr: longint;
    junk: integer;
    junk_icerr: ICError;
begin
  err := GotRequiredParams(theAppleEvent);
  if err = noErr then begin
    junk_icerr := ICGetPrefStr(instance, kICEmail,
                               junk_attr, email_address);
    ParamText(email_address, '', '', '');
    junk := Alert(128, nil);
    quit_now := true;
  end; (* if *)
  HandleOpenApplication := err;
end; (* HandleOpenApplication *)
```

HandleOpenDocuments

```
function HandleOpenDocuments (theAppleEvent:AppleEvent;
            reply: AppleEvent;
            refcon: longint): OSErr;
  (* a fairly standard 'odoc' event handler *)
  (* gets the document list, counts the items in it *)
  (* gets the FSSpec for each document and calls *)
  (* ProcessDocument on it *)
  var
    fss: FSSpec;
    doc_list: AEDescList;
    index, item_count: longint;
    junk_size: Size;
    junk_keyword: AEKeyword;
    junk_type: descType;
    err, junk: OSErr;
begin
  err := AEGetParamDesc(theAppleEvent, keyDirectObject,
            typeAEList, doc_list);
  if err = noErr then begin
    err := GotRequiredParams(theAppleEvent);
    if err = noErr then begin
      err := AECountItems(doc_list, item_count);
    end
    else begin
      item_count := 0;
    end; (* if *)
    for index := 1 to item_count do begin
      if err = noErr then begin
        err := AEGetNthPtr(doc_list, index, typeFSS,
                  junk_keyword, junk_type,
                  @fss, sizeof(fss), junk_size);
        if err = noErr then begin
          err := ProcessDocument(fss);
        end; (* if *)
      end; (* if *)
    end; (* for *)
    junk := AEDisposeDesc(doc_list);
  end; (* if *)
  HandleOpenDocuments := err;
end; (* HandleOpenDocuments *)
```

HandleQuit

```
function HandleQuit (theAppleEvent:AppleEvent;
            reply: AppleEvent;
            refcon: longint): OSErr;
  (* a fairly standard 'quit' event handler *)
  (* sets quit_now so that the main event loop quits *)
  var
    err: OSErr;
begin
  err := GotRequiredParams(theAppleEvent);
  if err = noErr then begin
    quit_now := true;
  end; (* if *)
  HandleQuit := err;
end; (* HandleQuit *)
```

Main Program

```
  var
    junkbool: boolean;
    event: EventRecord;
    err: OSErr;
    junk: OSErr;
    response: longint;
    attr: longint;
begin
  (* First check for System 7. OK, so we're supposed *)
  (* to test for functionality but this is example code. *)
  if (Gestalt(gestaltSystemVersion, response) <> noErr)
        | (response < $700) then begin
    ExitToShell;
  end; (* if *)

  (* Now install our AppleEvent handles. *)
  err := AEInstallEventHandler(kCoreEventClass,
            kAEOpenApplication,
            @HandleOpenApplication, 0, false);
  if err = noErr then begin
    err := AEInstallEventHandler(kCoreEventClass,
            kAEOpenDocuments,
            @HandleOpenDocuments, 0, false);
  end; (* if *)
  if err = noErr then begin
    err := AEInstallEventHandler(kCoreEventClass,
            kAEQuitApplication,
            @HandleQuit, 0, false);
  end; (* if *)

  (* startup Internet Config *)
  if err = noErr then begin
    err := ICMapErr(ICStart(instance, my_creator));
    if err = noErr then begin
      err := ICMapErr(ICFindConfigFile(instance, 0, nil));
    end; (* if *)

  (* fetch the mappings preference *)
    if err = noErr then begin
      err := ICMapErr(ICGetPrefHandle(instance, kICMapping,
                attr, mappings));
    end; (* if *)

  (* enter main loop *)
    if err = noErr then begin
      quit_now := false;
      while not quit_now do begin
        junkbool := WaitNextEvent(everyEvent, event,
                        maxlongint, nil);
        case event.what of
          keyDown:
            quit_now := true;
          kHighLevelEvent:
            junk := AEProcessAppleEvent(event);
          otherwise
            ;
        end; (* case *)
      end; (* while *)
    end; (* if *)

    (* shut down IC, only if we successfully started it *)
    junk := ICStop(instance);
  end; (* if *)

  (* beep if we get any errors*)
  (* sophisticated error handling this is not *)
  (* a good place to put a breakpoint this is *)
  if err <> noErr then begin
    SysBeep(10);
  end; (* if *)
end. (* SpaceAliens *)
```

This program, "Space Aliens Ate My Icons!", is a simple drag and drop utility that 'fixes' the type and creator of any file dropped on to it by looking up the file's extension in the IC mappings database. It combines a standard drag and drop application shell with some simple IC operations. There are four interesting sections of this program: the global variables, the main line, the HandleOpenApplication routine and the ProcessDocument routine.

1 Global Variables

There are two IC-related global variables in Space Aliens. The first, `instance`, is the standard connection to IC that all programs must obtain before they use IC. The variable is global so that it can be accessed throughout the program.

Although it is possible to start and stop IC multiple times during the execution of your program, it is normally easier to just start it at the beginning and stop it at the end.

The other IC-related global variable is `mappings`, which is a handle to the mappings database. More on this later.

2 Main Program

The main program does all of the things normally associated with the main line of an application, including checking the system configuration, installing AppleEvent handlers and running the main loop, but it also performs a number of IC-related operations. These include starting IC, finding a preferences file and stopping IC. These operations are done in almost the same way as in ICHelloCruelWorld; the only change is that Space Aliens has a proper signature and passes that to ICStart instead of `'????'`. IC currently ignores this parameter but it is important that applications pass their signature to this routine to support future extensions.

The other IC-related section of the main program reads the mappings database into the mappings global variable. This is done by calling ICGetPrefHandle, which is very similar to ICGetPref, but returns the result in a new handle. The main program stores this handle in a global variable so that ProcessDocument can use it to look for matching extensions.

The ICGetPrefHandle is not a IC API routine but instead is provided in a library, ICSubs. Its implementation is interesting because it highlights some useful features of the ICGetPref. The implementation is given in Listing 3.

Listing 3: ICSubs.p (extract)

ICGetPrefHandle

```
function ICGetPrefHandle (inst: ICInstance; key: Str255;
          var attr: ICAttr; var prefh: Handle): ICError;
  var
    err: ICError;
    prefsize: longint;
begin
  prefh := nil;
  prefsize := 0;
  err := ICGetPref(inst, key, attr, nil, prefsize);
  if err <> noErr then begin
    prefsize := 0;
  end; (* if *)
  prefh := NewHandle(prefsize);
  err := MemError;
  if err = noErr then begin
    HLock(prefh);
    err := ICGetPref(inst, key, attr, prefh^, prefsize);
    if err = icPrefNotFoundErr then begin
      attr := 0;
      err := noErr;
    end; (* if *)
    HUnlock(prefh);
  end; (* if *)
  if err <> noErr then begin
    if prefh <> nil then begin
      DisposeHandle(prefh);
    end; (* if *)
    prefh := nil;
  end; (* if *)
  ICGetPrefHandle := err;
end; (* ICGetPrefHandle *)
```

Notice that it first calls ICGetPref with a nil data pointer. ICGetPref then returns the current size of the preference in prefsize but doesn't actually return the data. Next ICGetPrefHandle creates a handle of the appropriate size, locks it and then calls ICGetPref again, this time with the dereferenced handle as the destination for the data. This handle is then returned to the calling program.

3 HandleOpenApplication

The HandleOpenApplication routine displays the program's about box, including (for no readily apparent reason) the user's Email address. This is a classic application of Internet Config to return simple preferences, in this case a Pascal string.

Note that ICGetPrefStr is not a IC API routine, but instead is provided in a library, ICSubs. Its implementation is very boring.

Listing 4: ICSubs.p (extract)

ICGetPrefStr

```
function ICGetPrefStr (inst: ICInstance; key: Str255;
            var attr: ICAttr; var str: Str255): ICError;
  var
    err: ICError;
    size: longint;
begin
  size := 256;
  err := ICGetPref(inst, key, attr, @str, size);
  if err <> noErr then begin
    str := '';
  end; (* if *)
  ICGetPrefStr := err;
end; (* ICGetPrefStr *)
```

4 ProcessDocument

This routine is called by the HandleOpenDocuments AppleEvent handler for each document in the direct parameter. ProcessDocument walks the mappings data structure looking for the entry with suitable flags and the longest extension that matches the file name. If it finds a matching entry, it sets the type and creator of the file to the values stored in the entry.

The structure of the mappings database is quite complicated, so IC provides another library, ICMappings, containing routines to parse it. The mappings database can be viewed as a big array of entries, each entry having the structure shown in Listing 5.

Listing 5: ICKeys.p (extract)

ICMapEntry

```
ICMapEntry = record
    total_length: integer; (* from beginning of record *)
    fixed_length: integer; (* from beginning of record *)
    version: integer;
    file_type: OSType;
    file_creator: OSType;
    post_creator : OSType;
    flags: longint;
    (* variable part starts here *)
    extension: Str255;
    creator_app_name: Str255;
    post_app_name : Str255;
    MIME_type: Str255;
    entry_name: Str255;
  end;
```

The thing to note about this record is that it contains five Str255s, making it approximately 1.25 KB. There are hundreds of these records in the mappings database, which makes for quite a large preference! Obviously this would be a problem, so the mappings database is compressed, with the Str255s being laid out sequentially in memory. Given that most of these strings will typically be short, this represents a significant memory saving.

This has the disadvantage that it makes the data structure difficult to parse. The task is further complicated by the fact that applications can append arbitrary data after the strings, using a protocol defined in the IC programming documentation. For this reason it is a good idea to use the routines in ICMappings to access the database.

So there you go, a simple and yet useful Internet Config aware application in under 300 lines of code.

Adding IC Support to Your Application

The two previous sections have covered most of the important things that you need to know to add Internet Config support to your application. This section contains a checklist for making your application IC aware:

- The first thing is… read the documentation! IC comes with comprehensive programming documentation that covers a lot more ground than this article can.

- Add ICGlue to your project. There are two libraries, one for 68K and one for PPC. Certain development environments require you to convert the 68K object file before you can use it, but this process is straightforward.

- Call ICStart at the beginning of your program and ICStop at the end.

- Call ICFindConfigFile, usually with the default parameters. If you implement double-clickable preference files then read the next section about a gotcha associated with them.

- Call ICGetPref when you need the value of a preference from IC. The list of these preferences is given in the programming documentation, but you can get a good idea of what is available by looking through the Internet Config application. If you are reading a lot of preferences at once, then you can make your program faster by bracketing your calls with calls to ICBegin and ICEnd.

- Remove any superfluous user interface for preferences that are better managed by IC.

If your program is small, you might want to discard your existing preference mechanism and use IC to store all of your preferences, including your private ones. The programming documentation gives details on how to store private preferences using IC.

Internet Config Gotchas

All system software has gotchas and IC is no exception. But "forewarned is forearmed", so here is my list of potential pitfalls.

Double-Clickable Preference Files

Many applications implement double-clickable preference files so that multiple users can share the same Macintosh. If your application supports this, you need to make sure that you are using Internet Config 1.1. The details of the problem are beyond the scope of this article but suffice to say that, due to a design error, version 1.0 does not provide sufficient support for double-clickable preference files. The issue is discussed in detail in the programming documentation. [See, I told you it was important to read the documentation!]

ICError

ICError is a longint (or long for you C people) – it is not an integer (short) or an OSErr. Not recognising this fact could cause all sorts of problems in your code. There is a routine in ICSubs called ICMapErr which converts ICErrors to OSErrs.

User Interface Frowned Upon

I recommend that you do not provide a user interface for changing the preferences that are managed by Internet Config. I do however recognise that some developers will not heed this recommendation, so the API provides support for your programs to change preferences – I merely suggest that you not use it.

Version 1.1 of IC provides an API call to launch the Internet Config application, bring it to the front and display a specific preference. In many cases this is all the user interface you need.

Read-Only Preferences

One of the bits in the preference attributes is a read-only bit. If this bit is set, then any attempt to write the preference will result in an error. If you provide a user interface for changing IC preferences, then you should make sure that you pay attention to this read-only bit. Any future version of IC that renders your user interface obsolete will also mark the affected preferences as read-only so that you don't attempt to change them. It is important that you indicate read-only preferences in your user interface, otherwise your users will get very confused as to why their preference changes are bouncing.

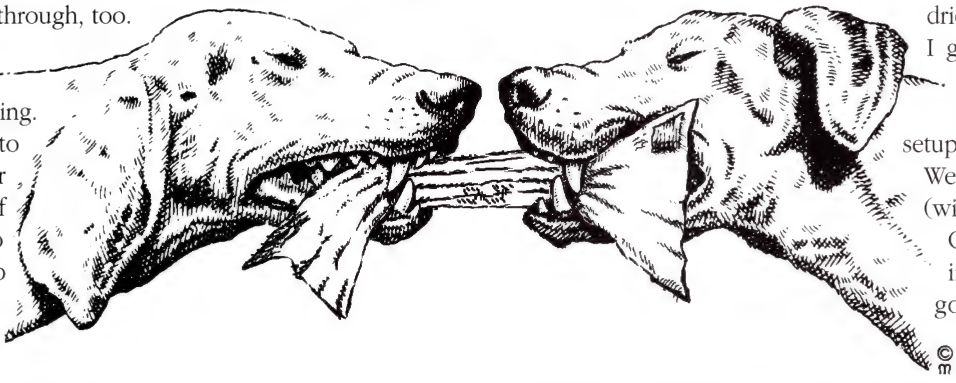
Continued on page 38

A SERIOUS SYSTEM ERROR HAS OCCURRED

By the much-chagrined Scott T Boyd

MacTech Loses Several Bags of E-mail!

We don't quite know how to say this. How about, "The dog ate it?" If you sent mail to editorial@xplain.com in the past few months, there's a pretty good chance that we never received it. There's also some chance that mail to other xplain.com addresses didn't get through, too.



We feel terrible about possibly having left you hanging. We try pretty hard to answer all of our mail. Generally, if we don't manage to send a response to an e-mail, it's the occasional human error. However, in this case, it's a wholesale failure.

If you sent something and never heard back from us, we ask that you please send it again.

The morbidly curious generally want to know what went wrong, so here's the deal. We use a UUCP gateway for QuickMail, and it picks up mail from our Internet provider, as well as delivers outgoing Internet mail.

I live and work in Northern California, so the mail server is set up to forward mail back out onto the Internet to my machine. We used QuickMail's forwarding capabilities to redirect mail from my QuickMail account to my Internet account. That had the unfortunate effect of readdressing all of that mail to look as if it had originally been sent by my QuickMail account. I could only figure out who sent it by opening the message and reading the RFC822 mail header. If I wasn't really careful, I would sometimes ignore such mail because I thought it was something I had sent to someone and cc'd myself on for some reason.

Please consider resending anything you may have sent to editorial@xplain.com any time in the past few months

In a moment of inspiration, one of our LA office staff figured out that we could change the UUCP gateway configuration to have it redirect incoming editorial@xplain.com mail to my machine. It seemed like a great idea. It would get mail to me faster, and it would leave the From: field intact.

Now, if this were the only source of e-mail for me, I would have noticed right away that the steady stream of e-mail had dried up. Unfortunately, I get mail from a lot of sources, and I never noticed that this new setup wasn't working.

We just figured this out (with many thanks to Ed Cessna, an astute and intrepid soul). We've gone through the mail logs looking for clues. What a mess! Our Star*Nine UUCP gateway lost its mind somewhere along the line. We don't know if it's the fault of the gateway, our Internet provider (Netcom), or some other undetermined cause. At any rate, we extracted a slew of e-mail addresses and sent out a note to them to ask them to resend any mail that might have gotten lost.

I'm frequently read the comp.risks newsgroup, so I feel obliged to point out that this really comes down to human error. We should have verified that the change worked. We could have noticed sooner that it hadn't. We definitely should have looked at the logs sooner. Even though the logs were confusing and nearly indecipherable, we would have noticed that something was amiss. We trusted the technology without ever verifying that it was doing what we thought – another lesson about risk learned the hard way.

We've gone back to the old, slow, sender-hiding way, but it's getting the mail through.

If you fell victim to this, please accept our apologies. We hope that you'll resend anything if you're unsure whether it got through.

APPLE WORLDWIDE DEVELOPERS CONFERENCE

A FOUNDATION FOR CHANGE

See the future at Apple's WWDC. Who should attend? Software Engineers. Consultants. Multimedia Developers. Systems Integrators. Technical Managers.

Get more information via Fax-on-Demand at 800–770–4863. (Outside the U.S., please call 415–637–2607 from your fax machine handset.) Or via the Worldwide Web: http://wwdc.carlson.com

Save $100

Register by April 10 and pay just $995

SAN JOSE CONVENTION CENTER

MAY 8–12, 1995

Internet Configuration System continued from page 35

Why No User Interface? Well, for a start, it is more work for you, and one of the goals of IC was to reduce your workload. But the real reason is that future extensions to IC may render your user interface superfluous, or perhaps even counter-productive.

For example, if a future version of IC supports application-specific preferences, then your user interface would become obsolete because it would not have a way of specifying whether changes made by the user affect the application specific preference or the global default preference.

An even better example is the Internet Config Random Signature extension. This is an override component which returns a random signature each time a client asks for the signature preference. If your application provides a user interface for changing the signature, then the presence of this component renders it useless. For this reason, the Random Signature component marks the signature preference as read-only and returns an error when you attempt to set it.

Don't Cache

IC essentially represents a shared database of preferences. This means that the value of a preference can change at any time. For this reason, it is important that you do not cache the value of a preference for any length of time. We recommend that you fetch the preference value every time that you need to use it.

It is possible that your current code base makes this awkward, in which case IC allows you to watch for preference changes. See the programming documentation for details.

One exception to this rule is the Mappings preference. This preference is large, expensive to fetch and expensive to parse. If you need to use it often, then you should probably cache the preference value. In this case you should make sure to read the documentation to learn about how to watch for changes to this preference.

The observant reader will notice that the Space Aliens program completely ignores the issue of caching. It can get away with this because, after the initial open document events have been processed, it quits and it always refetches the mappings preference when it is launched.

Under the Bonnet

Translator's Note: In Australia, the cover to the engine compartment of a car is called a bonnet. A hood is the cover to the passenger compartment on a convertible. Oh yeah, and the luggage compartment is called the boot. Which means that all of those elephant jokes go right over our heads.

At first glance IC appears to be a simple set of library routines that use the Resource Manager to access preferences stored in a file. However IC is more complicated than that. The ICGlue file which you link to your application contains three parts: the switch glue, the link-in implementation and the component glue. The relationship between these is shown in Figure 3.

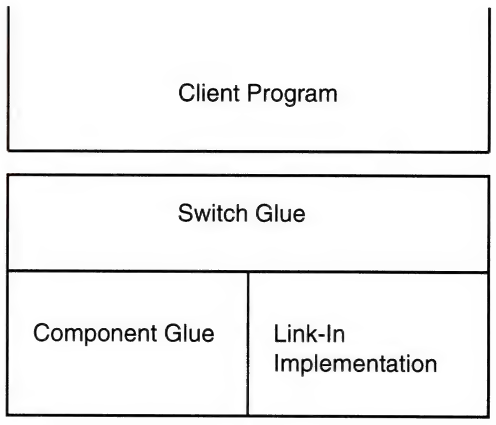


Figure 3. Structure of ICGlue

When you call ICStart, the switch glue attempts to open the Internet Config component. If this succeeds, then any other calls you make to IC are routed directly to the component. If ICStart fails to make a connection to the IC component, all subsequent calls to IC are routed to the link-in implementation. This code implements the basic functionality of IC using the Resource Manager.

Does this dialogue sound familiar?

Quinn: [very enthusiastically] We should do it using components!

Peter: [perplexed] What's a component?

Components are a lightweight form of dynamic linking which were originally designed by the QuickTime team, but have now escaped into the wider MacOS. Check out Inside Macintosh:More Macintosh Toolbox wherein the Component Manager is documented properly.

This scheme has a number of important benefits. Firstly, any program using IC can call it without having to worry about whether the component is installed. Secondly, the link-in implementation works with any system (well, at least back to System 6) which means IC can be used by any modern application without requiring the Component Manager or any other shared library mechanism. But the biggest benefit by far is that, if the component is present, then all calls are routed to the component; it acts as a dynamically linked library. So if we discover a bug in the link-in implementation (or the implementation becomes obsolete because of changes to the MacOS), we can replace the IC component and all old applications will benefit from the new implementation without recompilation.

Call 1-800-282-2732 to order today

You Can't Buy This CD in any Store

More than 30 popular SDKs—one low price

What's the easiest and least costly way to make the most of the new features enabled by Macintosh system software extensions?

In a word (or five), the Mac™ OS Software Developer's Kit (SDK).

The Mac OS SDK helps you support the entire Mac OS family, by offering instant access to *more than 30* of Apple's most proven and popular SDKs. And whereas these SDKs previously sold separately for nearly $2,000, now you can have them all on one CD—for a one-year subscription price of only $299.

You won't find this special Mac OS SDK in any store or catalog. It's available *only* from APDA, Apple's source for development tools. Nor will you be left on your own as soon as you make the purchase. Every quarter you'll receive an automatic update—ensuring you always have the most current information as well as an understanding of how to add support for advanced system extensions in your software.

A typical SDK provides the system software extension; programming interfaces and libraries; sample code; and technical documentation. Here's a listing of the SDKs that are included in your first CD mailing:

Apple Guide
Apple Open Collaboration Environment (AOCE)
Apple Remote Access
Apple Remote Access Modem
Apple Shared Library Manager
AppleScript
AppleSearch
AppleShare API
AppleTalk Wide Area
ColorSync
Communications Toolbox
Control Strip
Designing PCI Cards & Drivers
File System Manager
Installer
Macintosh Drag and Drop
Macintosh Easy Open
MacODBC
MacOSI Connection
MacSNMP
MacTCP
MacX.400
MacX25
MIDI Management Tools
Network Software Installer
Open Transport
PlainTalk
QuickDraw GX
QuickTime
Sound Manager
Telephone Manager
Thread Manager
XTND

How to Order Mac OS SDK

To order by phone: call **1-800-282-2732** (U.S.) **1-800-637-0029** (Canada), or **(716) 871-6555** (International) Monday through Friday, 7am to 5pm PST.

R0603LL/A $299 per annual subscription (four mailings).

To order by mail: send a check or money order for $299 payable to APDA/Apple Computer, Inc., plus sales tax and $20 for shipping and handling (for U.S. orders) to: APDA, Apple Computer, Inc., P.O. Box 319, Buffalo, NY 14207-0319. For international orders, please call APDA for shipping and handling charges. Allow 5-10 days for delivery within the U.S.

MTM0495

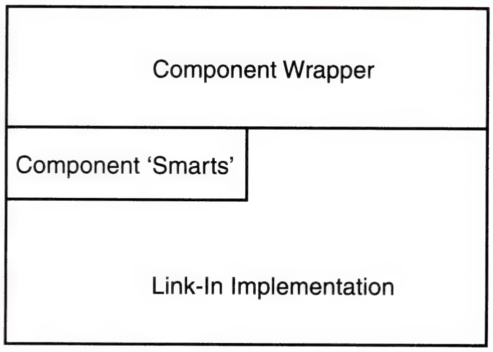


Figure 4. Structure of the IC component

The IC component, whose structure is shown in Figure 4, uses exactly the same code as the link-in implementation used by the ICGlue. Layered on top is the component wrapper, which provides the support required by the Component Manager. In addition, the component 'smarts' are sandwiched between the component wrapper and the link-in implementation. As we develop IC, the component is getting smarter than the link-in implementation, providing improvements in internationalisation, efficiency and so on.

Finally, the link-in implementation is a pretty ordinary preference file implementation. The preferences are stored as 'PREF' resources in the Internet Preferences file in the Preferences folder. The resource name is used as the preference key and the first four bytes of the resource hold its attributes. The rest of the resource holds the preference's value.

Oh, by the way, the full source code to all parts of the Internet Config System is in the public domain. So, if you're worried about the quality of our code, you can take a look for yourself.

The Future

Internet Config can be extended in a number of ways. The first and most obvious mechanism for extending it is to release new versions of the Internet Config component. As we find bugs in IC, we will release new versions of the component that fix them – this is the raison d'être for components.

The second way of extending Internet Config is by using override components. The Component Manager has this cool ability to let one component capture another component and then selectively pass calls through to the captured component. This mechanism, very much like inheritance in object oriented design, allows very simple components to be written that capture the Internet Config component to patch just one routine. An example of this is the Internet Config RandomSignature extension. This component captures the Internet Config component and overrides the ICGetPref routine so that, if the call is requesting the signature preference, it returns a random signature instead of the one stored in the Internet Preferences file.

The possibilities for override components are endless. For example, let's say your organisation wants to pre-configure all of its news clients to access the central news server. All you need to do is write a simple override component that watches for programs getting the NNTPHost preference and return a fixed value as a read-only preference. This way all news readers will use the correct host and the users won't be able to change it. As we say in the system software business... wonderful third party developer opportunity.

The third way of extending IC is by replacing the entire Internet Configuration System. If you replace both the Internet Config component and the Internet Config application, then you have total control over all aspects of the system. For example, imagine you want a version of IC that implements application-specific preferences. The first step would be to replace the component with a smarter component, capable of storing a set of preferences for each application and returning the right preferences to the right application. Then you would replace the Internet Config application with a more sophisticated application which can manage multiple sets of preferences. Your job is done – all IC aware programs will automatically benefit without recompilation.

"IC is a very flexible system and I look forward to seeing it extended in ways I never anticipated."

As one final example, suppose you want to store your user preferences on a central server and access them through some network protocol. IC provides the flexibility to do this by replacing the standard component with a network-aware one. Of course, you would have to work out the user's identity in some way, perhaps by requiring them to log on before using any IC aware programs. You can then choose whether to write a Macintosh application to administer the server or use tools from the server's native environment.

IC is a very flexible system and I look forward to seeing it extended in ways I never anticipated.

Conclusion

The Internet is a cruel place for users. When a Macintosh user ventures onto the net, things that they take for granted, such as file types and the Chooser, suddenly stop working. Fortunately, the Macintosh offers superb Internet tools to ease that transition. Nevertheless, users need as much help as they can get, and one great way to help your users is to reduce the number of preferences they have to deal with – and the best way to do that is to support Internet Config.

Internet Config can be called from all common Macintosh development environments. Internet Config is easy to program and you can add it to your existing application with a minimum of fuss. Internet Config allows you to eliminate user interface code from your program, making it smaller and easier to maintain. Internet Config has full source code available in the public domain, so you can base commercial solutions around it without worrying about a single vendor hijacking your destiny. And finally, Internet Config is supported by a wide array of commercial, freeware and shareware Internet tool developers.

So there really is no excuse. Why aren't you coding already???

Further Reading

If you want to find out more about IC and the technology that it is based upon then the following documents may be of interest.

Quinn, *Internet Configuration System: User Documentation*, 1994

Quinn, *Internet Configuration System: Programming Documentation*, 1994

Apple Computer, Inc, *Inside Macintosh: More Macintosh Toolbox*, 1993, Addison-Wesley

QuickTime Team & Dave Radcliffe, "QT 05 – Component Manager version 3.0", *New Technical Notes*, Mar. 1994, Apple Developer Support

David Van Brink, "Be Our Guest: Components and C++ Classes Compared", **develop**, Issue 12, Dec. 1992, Apple Developer Press

Bill Guschwan, "Inside QuickTime and Component-Based Managers", **develop**, Issue 13, Mar. 1993, Apple Developer Press

John Wang, "Somewhere in QuickTime: Derived Media Handlers", **develop**, Issue 14, June 1993, Apple Developer Press

Gary Woodcock, "Managing Component Registration", **develop**, Issue 15, Sep. 1993, Apple Developer Press

DEVELOPER
DU
UNIVERSITY

Power Up with PowerPC !

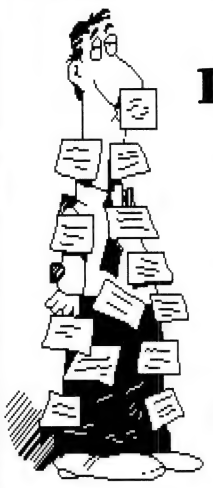
Apple's Developer University has the right courses to jump start your PowerPC development efforts:

- ❑ **Programmer's Introduction to RISC and PowerPC self-paced training**
- ❑ **PowerPC BootCamp Class**

And, with over 19 courses, we offer the fastest way to get up to speed on the Macintosh, whether you're just getting started or seeking to understand Apple's newest development technologies.

For more information, contact the Apple Developer University Registrar by telephone at (408) 974-4897 or fax (408) 974-0544.

Developer University, Apple Computer, Inc. 1 Infinite Loop, MS 305-1TU, Cupertino, CA 95014



DON'T FORGET!

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

POSITIONS WANTED

Available Immediately 1-800-736-3577

Expert 4D Programmers Less than 50¢ per hour!

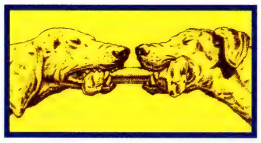
More than 1,000 hours of development went into 4D Toolkit 2.0. With a one time price of $395.00*, it's like hiring a crack team of 4D programmers at less than 50¢ per hour. Call 1-800-736-3577 to order your copy, or to receive a **free** demo.

4D TOOL**KIT**™

OPtions Computer Consulting
228 Bleecker Street #19
New York, NY 10014
TEL: 212-645-3577
FAX: 212-633-0336

* *upgrade pricing available*

PROGRAMMER'S CHALLENGE



By Mike Scanlin, Mountain View, CA

Stock Market Database

This month's Challenge is to write a piece of code that records stock market trades and then allows you to query it to find out price and volume information for a particular stock at a particular time. This code could be the core of a much larger stock analysis program (and it would likely be the bottleneck).

The typedefs you will use are:

```
typedef unsigned char uchar;
typedef unsigned long ulong;

typedef struct TimeStamp {
   uchar       yearsFrom1900;
   uchar       month;
   uchar       day;
   uchar       hour;
   uchar       minute;
   uchar       second;
} TimeStamp;

typedef char Str7[8];

typedef struct Trade {
   Str7       symbol;
   TimeStamp  time;
   Fixed      price;
   ulong      numShares;
} Trade;
```

The four routines you'll write are:

```
void *
InitTradeDatabase(maxRAM)
ulong       maxRAM;

void
NewTrade(privateDataPtr, trade)
void        *privateDataPtr;
Trade       trade;

Fixed
PriceIs(privateDataPtr, symbol, time);
void        *privateDataPtr;
Str7        symbol;
TimeStamp   time;
ulong
VolumeIs(privateDataPtr, symbol, time);
void        *privateDataPtr;
Str7        symbol;
TimeStamp   time;
```

InitTradeDatabase is called once before any other functions. It is untimed (i.e. it doesn't matter if it's slow to execute) and should return a pointer to your database's private data (which is passed to the other 3 routines as privateDataPtr). MaxRAM is the largest amount of RAM that your code can use (in bytes); it will be between 512K and 4MBs. You can also use up to 50MBs of disk space.

Once InitTradeDatabase has been called, the other 3 routines will be called pseudo-randomly many times. The only restriction is that each time NewTrade is called, the trade's time will be later than all previous trades. You can think of NewTrade being called once for each trade that occurs, as it occurs (like the data flowing across a ticker tape, which happens in chronological order).

The chronological sequence of NewTrade calls will be interspersed with calls to PriceIs and VolumeIs. PriceIs returns the price of a given stock at or before the given time. If you are asked for the price of a stock at a time before you have any trade data for that stock then return a price of zero.

VolumeIs returns the daily volume of a given stock as of a given time on that day. For example, if the time is 2/28/95 at 11am then VolumeIs should return the sum of all trades' numShares that occurred on the 28th of February, 1995, before 11am (and excluding trades that occurred at exactly 11am).

Taken together these routines will allow someone to produce price/volume graphs for a stock of their choice (once they've fed it lots of trade data).

Here are some examples. A time of 13:32:15 (15 seconds past 1:32pm) on Mar 2nd, 1995, is:

```
yearsFrom1900 = 95;
month = 3;
day = 2;
```

THE RULES

Here's how it works: Each month we present a new programming challenge here. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech Magazine. We choose a winner based on code correctness, speed, size and elegance (in that order of importance) as well as the postmark of the answer. In the event of multiple equally-desirable solutions, we'll choose one winner at random (with honorable mention, but no prize, given to the runners up). The prize for each month's best solution is $50 and a limited-edition "The Winner! MacTech Magazine Programming Challenge" T-shirt (not available in stores).

To help us make fair comparisons, all solutions must be in ANSI compatible C (e.g. don't use Think's Object extensions). Use only pure C code. We disqualify any entries with any assembly in them (except for challenges specifically stated to be in assembly). You may call any Mac routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We test entries with the FPU and 68020 flags turned *off* in THINK C. We time routines with the latest THINK C (with "ANSI Settings", "Honor 'register' first", and "Use Global Optimizer" turned on), so beware if you optimize for a different C compiler. **Limit your code to 60 characters wide**; this helps with e-mail gateways and page layout.

We publish the solution and winners for this month's Programmers' Challenge two months later. All submissions must be ***received by*** the 10th day of the month printed on the front of this issue.

You can get a headstart on the challenge by reading the online version. We post it to the online services at the same time that we post source code. We'll make every effort to have it online no later than when the magazines get mailed out, but we're unable to guarantee that it will be online by any given date.

Mark solutions "Attn: Programmers' Challenge Solution" and send them by e-mail – *Internet* progchallenge@xplain.com, *eWorld* MT.PrgChal, *AppleLink* MT.PROGCHAL, *CompuServe* 71552,174 and *America Online* MT PRGCHAL. Include the solution, all related files, and your contact info. For more details, see "How to Contact Us" on p. 2.

MacTech Magazine reserves the right to publish any solution entered in the Programming Challenge of the Month. Authors grant MacTech Magazine the non-exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

```
    hour = 13;
    minute = 32;
    second = 15;
```

A price of 14 and 11/16ths would be:

```
    price = 0x000EB000; /* 14.6875 */
```

Remember, a Fixed is 16 bits of integer (0x000E) and 16 bits of fraction (0xB000). You can think of it as a 32 bit integer that you could divide by 2^{16} to get the floating point equivalent. The value 1 is 0x00010000. The value 0.5 is 0x00008000. Stock prices are normally quoted in 1/2s, 1/4s, 1/8s, 1/16ths, 1/32nds and 1/64ths, and those are the only possible fractions your code will receive.

Symbols are uppercase, 7 character PStrings. The symbol for Apple Computer is AAPL. It would be:

```
Str7 symbol;
symbol[0] = 4; /* length */
symbol[1]='A'; symbol[2]='A'; symbol[3]='P'; symbol[4]='L';
```

NumShares is always greater than zero and will only very rarely be larger than 100,000 (the max, for our purposes, is 10,000,000).

Note that, on a typical day, the stock exchanges of the world have hundreds of thousands of transactions. It is all but certain that your routine will run out of RAM. You will need to be prepared to swap some trade data to disk. And then you need an efficient way to retrieve that data if you are asked for price or volume information once you've swapped it to disk. Part of this Challenge is to come up with a clever way to store RAM indexes of disk-based trade data. And you'll probably want to cache at least some (if not all) of the trade data in RAM when you can. You will not be given more than 10MB of trade data (since you can use 50MB of disk space, you should be fine).

Write to me if you have any questions. Happy trading.

Two Months Ago Winner

Congratulations to **Gustav Larsson** (Mountain View, CA) for winning the Symbolize Challenge. Last month, Gustav was complimented for his small code that was only slightly slower than the winner. This month he has the largest code, but it's almost 3x faster than the nearest competitor.

Here are the times and code sizes for each entry. Numbers in parens after a person's name indicate that person's cumulative point total for all previous Programmer Challenges, not including this one (see Top 20 chart below for info on the new cumulative point total plan):

Name	time	code
Gustav Larsson (10)	74	2942
Paul Hoffman	197	2100
Scott Manjourides	205	1972
Mason Thomas	235	1742
David Salmon	239	946
Dave Darrah (26)	249	1392
David Wiser	355	1492

Gustav's solution is extremely nice code. It's well thought out (good algorithms), well implemented (he knows his compiler) and well commented. If all of the authors of my favorite applications took as much care at crafting their code, then I'm sure I'd get at least an extra half hour of work done each day. I highly recommend that you study his code and comments.

Top 20 Contestants of All Time

This month marks the 32nd installment of the Programmer's Challenge in MacTech. Prompted by Bob Boonstra's retirement in February, I decided I needed to have some way to recognize past entrants who had done well. Thus was born the Top 20 Of All Time Chart.

Here's how it works. There are three ways to earn points: (1) by scoring in the top 5 of any particular challenge, (2) by being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	5 points
suggesting challenge	2 points

Each month I will present the cumulative point totals for the top 20 contestants. It took me a while to compile the point totals for this month's chart. I may have made a mistake. If you think you deserve more points than I've given you, please write to me and explain why (i.e. give me a list the months you finished in the top 5 as well as the months you found a bug or suggested a challenge I used and I'll check it out). If your name is not in the current Top 20 but you want to know how many points you have, then e-mail me and I'll tell you. Note that the numbers below include points awarded for this months' top 5 entrants.

So, here it is. Congrats, everyone, on the hard work it took to get here! (Note: ties are listed alphabetically by last name -- there are 23 people listed this month because 7 people had 20 points each.)

Top 20 Contestants of All Time

1.	Boonstra, Bob	176
2.	Karsh, Bill	71
3.	Stenger, Allen	65
4.	Cutts, Kevin	56
5.	Riha, Stepan	51
6.	Goebel, James	49
7.	Munter, Ernst	48
8.	Nepsund, Ronald	40
9.	Vineyard, Jeremy	40
10.	Larsson, Gustav	30
11.	Landry, Larry	29
12.	Mallet, Jeff	27
13.	Darrah, Dave	26
14.	Elwertowski, Tom	24
15.	Kasparian, Raffi	22
16.	Lee, Johnny	22

17. Anderson, Troy	20
18. Burgoyne, Nick	20
19. Galway, Will	20
20. Israelson, Steve	20
21. Landweber, Greg	20
22. Noll, Bob	20
23. Pinkerton, Tom	20

Here is Gustav's winning solution:

SYMBOLIZE.C
Copyright © 1995 Gustav Larsson

/* Three areas of this program have been optimized: memory allocation, file I/O, and the symbol parse/convert algorithms. Memory allocation and file I/O together typically use up 70-80% of the total execution time. Tuning these areas is tricky because so much is out of our control. Factors such as the state of the disk cache and the heap can affect overall execution time by 10%, completely swamping many optimizations in the parse/convert algorithms. Still, I have heavily optimized the parse/convert algorithms because that was the fun part of writing the program. Limitations: - This program will not tolerate blank lines in the input file or symbol file (a final carriage-return is okay). - Symbols names longer than about 2K may sometimes cause the output buffer to be overrun, trashing memory. This seems like a more-than-reasonable limit, even for mangled C++ names. */

```
#pragma options(pack_enums)  /* required by <Memory.h> */

#include <stdio.h>
#include <Memory.h>
```

```
typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned long ulong;

#define CEILING(value,n) (n*(((value)+n-1)/n))
```

/* There is a substantial performance hit if a write exceeds the disk cache. Large buffers also take longer to allocate via NewPtr. Any buffer size between 8K and 16K seems to produce comparable results. The actual buffer size here is 16K, to allow the contents to run over the 14K threshold. */

```
#define OUTPUT_BUFSIZE (16*1024)
#define OUTPUT_THRESHOLD (14*1024)
```

ishex

/* ishex[] indicates which ASCII characters are valid hex digits (0-9, A-F, a-f). There are entries for all 256 character codes since anything could be in the input file.*/

```
static uchar ishex[] = {
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 00-0F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 10-1F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 20-2F */
  1,1,1,1,1,1,1,1, 1,1,0,0,0,0,0,0,   /* 30-3F */
  0,1,1,1,1,1,1,0, 0,0,0,0,0,0,0,0,   /* 40-4F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 50-5F */
  0,1,1,1,1,1,1,0, 0,0,0,0,0,0,0,0,   /* 60-6F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 70-7F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 80-8F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* 90-9F */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* A0-AF */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* B0-BF */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* C0-CF */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* D0-DF */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,   /* E0-EF */
  0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0    /* F0-FF */
};
```

hexFromChar

/* hexFromChar[] converts a single hex character to its binary value. Since this table is only used the character is known to be a hex digit, we can end the table at 'f'. */

```
static uchar hexFromChar[] = {
  0, 0, 0, 0, 0, 0, 0, 0,  0,0,0,0,0,0,0,0,   /* 00-0F */
  0, 0, 0, 0, 0, 0, 0, 0,  0,0,0,0,0,0,0,0,   /* 10-1F */
  0, 0, 0, 0, 0, 0, 0, 0,  0,0,0,0,0,0,0,0,   /* 20-2F */
  0, 1, 2, 3, 4, 5, 6, 7,  8,9,0,0,0,0,0,0,   /* 30-3F */
  0,10,11,12,13,14,15, 0,  0,0,0,0,0,0,0,0,   /* 40-4F */
  0, 0, 0, 0, 0, 0, 0, 0,  0,0,0,0,0,0,0,0,   /* 50-5F */
  0,10,11,12,13,14,15                         /* 60-66 */
};
```

charsFromInt

/* charsFromInt[] converts a binary value (0..99) to a pair of decimal digits. The ULTOA macro converts two digits at a time. */

```
static ushort charsFromInt[] = {
  '00','01','02','03','04','05','06','07','08','09',
  '10','11','12','13','14','15','16','17','18','19',
  '20','21','22','23','24','25','26','27','28','29',
  '30','31','32','33','34','35','36','37','38','39',
  '40','41','42','43','44','45','46','47','48','49',
  '50','51','52','53','54','55','56','57','58','59',
  '60','61','62','63','64','65','66','67','68','69',
  '70','71','72','73','74','75','76','77','78','79',
  '80','81','82','83','84','85','86','87','88','89',
  '90','91','92','93','94','95','96','97','98','99'
};
```

```
void Symbolize( FILE *inputFile, FILE *symbolFile,
                FILE *outputFile, unsigned short symLength);

static unsigned short parseSymbols( char *buffer,
                char **name, ulong *value );

static void convert( char *inputBuffer, char *outputBuffer,
             char **symbolName, ulong *symbolValue,
             ushort numSymbols, ushort symLength,
             size_t inputLength, FILE* outputFile );
```

Symbolize

```
void Symbolize( FILE *inputFile, FILE *symbolFile,
                FILE *outputFile, unsigned short symLength)
{
  char *memory, *inputBuffer, *symbolBuffer, *outputBuffer;
  char **symbolName;
  size_t inputMax, symbolFileMax, len;
  ushort maxSymbols, numSymbols;
  ulong *symbolValue;
```

/* Compute buffer sizes, making them as tight as possible. Round up for alignment. Up to 9 characters can be added to the input buffer ("\r00000000"). See convert(). Up to 2 characters can be added to the symbol file buffer ("\r\0"). The shortest possible line in the symbol file is 11 characters ("12345678 A\r"). Also, reserve room for one extra symbol since the last one gets duplicated; see parseSymbols(). */

```
  inputMax = CEILING( inputFile->len + 9, 4 );
  symbolFileMax = CEILING( symbolFile->len + 2, 4);
  maxSymbols = symbolFile->len / 11 + 1;
```

/* Grab all the memory in one call to NewPtr, then parcel it out. This one call usually takes about 50% of the total execution time, at least on my machine. */

```
  memory = NewPtr( inputMax + symbolFileMax + OUTPUT_BUFSIZE
         + (sizeof(char*) + sizeof(ulong*)) * maxSymbols );

  if ( !memory )
    return;  /* at least the machine won't crash */

  inputBuffer = memory;
  symbolBuffer = inputBuffer + inputMax;
  outputBuffer = symbolBuffer + symbolFileMax;
  symbolName = (char **)( outputBuffer + OUTPUT_BUFSIZE );
  symbolValue = (ulong *)( symbolName + maxSymbols );
```

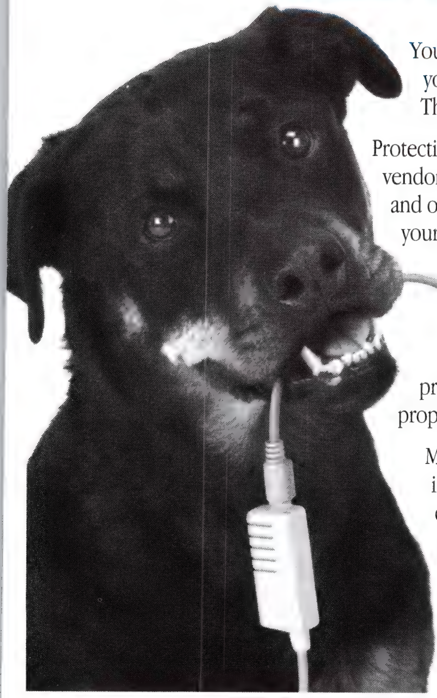
/* Put files into binary mode. This can reduce execution time by 20-40% for this program. Setting files to binary mode should really be done at fopen(). */

```
  inputFile->binary =
  symbolFile->binary =
  outputFile->binary = 1;
```

/* Read the symbol file. Add a final '\r' if necessary. Parse it, filling in symbolName[] and symbolValue[]. */

TASTES LIKE CHICKEN!

Without secure software, even the most expensive hardware key is nothing but a chew toy.



You need to protect your software. Hardware vendors will try to convince you that hardware keys are the only secure method of protection. They're wrong.

Protection is only as effective as the software involved. Most hardware key vendors require that you, the developer, spend major resources protecting and obscuring the code which interacts with a key. Without this effort on your part, your software is vulnerable, even to inexperienced hackers.

At PACE, we understand that software is the most important component of a working protection system. For more than a decade we've developed low cost and secure software based protection schemes. Our MacEncrypt system will turnkey protect your application automatically, applying multiple layers of PACE proprietary encryption and self checking algorithms to your product.

MacEncrypt is a secure, flexible, compatible and low cost protection investment. No false promises, wasted development time, upset users or expensive chew toys.

Trust the protection of your software to the people who understand software. Call today to order your PACE Developers Kit.

PACE Anti-Piracy 1082 Glen Echo Ave., San Jose, CA 95125
Vox: (408) 297-7444 • Fax: (408) 297-7441 • AppleLink: PACE.AP
email: info@paceap.com • Web page: http://paceap.com/pace.html

PACE
ANTI-PIRACY

"Our Japanese font supplier imposes strict copy-protection obligations on Adobe Systems. Because of this, we have been using PACE software protection on Adobe Type Manager,™ Japanese version and our other Japanese font products for almost 5 years. PACE's software solution provides us with effective security at a low cost. Our working relationship with PACE is excellent and their expert technical staff has always been helpful."

Paul Anderson
Senior Director, Pacific Rim
Adobe Systems, Inc.

```
  len = fread( symbolBuffer, 1, symbolFileMax, symbolFile );
  if ( symbolBuffer[ len-1 ] != '\r' )
    symbolBuffer[ len++ ] = '\r';
  symbolBuffer[ len ] = '\0';

  numSymbols = parseSymbols( symbolBuffer, symbolName,
                                          symbolValue );

 /* Read the input file.  Add a file '\r' if necessary.
   convert() will write the output file.*/

  len = fread( inputBuffer, 1, inputMax, inputFile );
  if ( inputBuffer[ len-1 ] != '\r' )
    inputBuffer[ len++ ] = '\r';

  convert( inputBuffer, outputBuffer,
           symbolName, symbolValue,
           numSymbols, symLength,
           len, outputFile );

 /* Deallocate memory.  */

  DisposePtr( memory );
}
```

```
/* I had a great deal of trouble getting Think C to store pointer variables in registers.
An address register would sometimes remain allocated even after it went out of scope
(this didn't happen with data registers).  In desperation, I created a single char*
register variable that gets passed into various macros.  When repeatedly accessing an
array, the first access below is slightly faster:
*
*   int array[10];
*   register int *parray = array;
*   foo = parray[i];  <- slightly faster
*   foo = array[i];
*
* Here are some casts for the shared address register: */

#define TEMP_UC ((uchar*)temp)
#define TEMP_US ((ushort*)temp)
#define TEMP_UL ((ulong*)temp)
```

XTOUL

```
/* XTOUL converts ASCII hex (X) to an unsigned long (UL).  Ptr gets bumped past the
8 hex digits. */

#define XTOUL(ptr,result,temp)     \
{                                  \
  register ulong _value;           \
                                   \
  temp = (char*) hexFromChar;      \
  _value = TEMP_UC[ *ptr++ ];      /* nybble 7 (high) */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 6 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 5 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 4 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 3 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 2 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 1 */ \
  _value <<= 4;                    \
  _value += TEMP_UC[ *ptr++ ];     /* nybble 0 (low) */ \
  result = _value;                 \
}
```

ULTOA

```
/* ULTOA converts an unsigned long (UL) to ASCII decimal (A) two digits at a time.  Buffer
should be an array of 5 ushorts and will be filled with the right-justified ASCII digits (2
chars per ushort).  Length will be set to the number of digits.  This code is optimized for
the common case of just a few digits.  / and % are inlined for 16-bit operands but generate
a function call for 32-bit operands (except on a 68020 or better). */

#define TEN_TO_THE_4th 10000
```

```
#define TEN_TO_THE_5th 100000
#define TEN_TO_THE_6th 1000000
#define TEN_TO_THE_7th 10000000
#define TEN_TO_THE_8th 100000000
#define TEN_TO_THE_9th 1000000000

#define ULTOA(value,buffer,length,temp)                    \
{                                                          \
  temp = (char*) charsFromInt;                             \
                                                           \
  if ( value < 100 )                                       \
  {                                                        \
    register ushort sval = value;                          \
    buffer[4] = TEMP_US[ sval ];                           \
    length = ( sval < 10 ? 1 : 2 );                        \
  }                                                        \
  else if ( value < TEN_TO_THE_4th )                       \
  {                                                        \
    register ushort sval = value;                          \
    buffer[4] = TEMP_US[ sval%100 ];                       \
    buffer[3] = TEMP_US[ sval/100 ];                       \
    length = ( sval < 1000 ? 3 : 4 );                      \
  }                                                        \
  else if ( value < TEN_TO_THE_6th )                       \
  {                                                        \
    register ushort sval = value % 10000;                  \
    buffer[4] = TEMP_US[ sval%100 ];                       \
    buffer[3] = TEMP_US[ sval/100 ];                       \
    buffer[2] = TEMP_US[ value/10000 ];                    \
    length = ( value < TEN_TO_THE_5th ? 5 : 6 );           \
  }                                                        \
  else if ( value < TEN_TO_THE_8th )                       \
  {                                                        \
    register ushort sval = value % 10000;                  \
    buffer[4] = TEMP_US[ sval%100 ];                       \
    buffer[3] = TEMP_US[ sval/100 ];                       \
    sval = value / 10000;                                  \
    buffer[2] = TEMP_US[ sval%100 ];                       \
    buffer[1] = TEMP_US[ sval/100 ];                       \
    length = ( value < TEN_TO_THE_7th ? 7 : 8 );           \
  }                                                        \
  else                                                     \
  {                                                        \
    register ushort sval = value % 10000;                  \
    buffer[4] = TEMP_US[ sval%100 ];                       \
    buffer[3] = TEMP_US[ sval/100 ];                       \
    sval = (value / 10000) % 10000;                        \
    buffer[2] = TEMP_US[ sval%100 ];                       \
    buffer[1] = TEMP_US[ sval/100 ];                       \
    buffer[0] = TEMP_US[ value/TEN_TO_THE_8th ];           \
    length = ( value < TEN_TO_THE_9th ? 9 : 10 );          \
  }                                                        \
}
```

LOOKUP

```
/* LOOKUP performs a binary search of valueTable[] and returns an index such that:
valueTable[index] <= value < valueTable[index+1]
valueTable[numSyms-1] and valueTable[numSyms] should both equal FFFFFFFF, so
that index will be numSyms-1 in this case. */

#define LOOKUP(value,index,valueTable,numSyms,temp) \
{                                   \
  register ulong _lo, _hi;          \
                                    \
  temp = (char *) valueTable;       \
  _lo = 0;                          \
  _hi = numSyms;                    \
  while ( _lo+1 != _hi )            \
  {                                 \
    index = (_lo + _hi) >> 1;       \
    if ( value < TEMP_UL[index] )   \
      _hi = index;                  \
    else                            \
      _lo = index;                  \
  }                                 \
  index = _lo;                      \
}
```

OUTPUT_SYMBOL

```
/* OUTPUT_SYMBOL writes "symbol+offset" into the output buffer.  outPtr is bumped
past the string.  Remember that a symbol name is terminated with a '\r', not '\0'. */

#define OUTPUT_SYMBOL(offset,symName,outPtr,temp) \
{                                                   \
  register ulong _offset;                           \
                                                    \
 /* Copy symbol name */                             \
  {                                                 \
    temp = symName;                                 \
    while ( ( *outPtr++ = *temp++ ) != '\r' ) ;     \
  }                                                 \
                                                    \
/* We've copied the \r into the output buffer.  \ If offset is nonzero, replace \r with + \
and output the offset (decimal).  If offset is zero, back up one char.  */           \
                                                    \
  _offset = offset;                                 \
  if ( _offset )                                    \
  {                                                 \
    ushort _buffer[5], _bufLen;                     \
    ULTOA(_offset,_buffer,_bufLen,temp)             \
    {                                               \
      temp = ((char*) &_buffer[5]) - _bufLen;       \
      *(outPtr-1) = '+';                            \
      while ( _bufLen-- )                           \
        *outPtr++ = *temp++;                        \
    }                                               \
  }                                                 \
  else outPtr--;    /* offset is zero */         \
}
```

OUTPUT_ADDRESS

```
/* OUTPUT_ADDRESS write "[symbol+offset  ]" into the output buffer.  outPtr is
bumped past the string.  There will be exactly symLen characters between "[" and "]".
Truncate the symbol name or pad with spaces as necessary.  _count is the number of
characters remaining.  _trunc is the number of characters remaining in the symbol
name (possibly truncated).  It gets reused in the second for loop to hold the number of
digits remaining in the decimal offset. */

#define OUTPUT_ADDRESS(offset,symName,symLen,outPtr,temp) \
{                                                          \
  register ulong _offset = offset;                         \
  if ( _offset )                                           \
  {                                                        \
    ushort _buffer[5], _bufLen;                            \
    ULTOA(_offset,_buffer,_bufLen,temp)                    \
    {                                                      \
      register int _count, _trunc;                         \
      *outPtr++ = '[';                                     \
      for ( temp = symName, _count = symLen,               \
                _trunc = _count-_bufLen-1;                 \
            _trunc && *temp != '\r';                       \
            _count--, _trunc-- )                           \
        *outPtr++ = *temp++;                               \
                                                           \
      *outPtr++ = '+';                                     \
      _count--;                                            \
      for ( temp = ((char*) &_buffer[5]) - _bufLen,        \
                _trunc = _bufLen;                          \
            _trunc;                                        \
            _count--, _trunc-- )                           \
        *outPtr++ = *temp++;                               \
      while ( _count-- )                                   \
        *outPtr++ = ' ';                                   \
      *outPtr++ = ']';                                     \
    }                                                      \
  }                                                        \
  else                                                     \
  {                                                        \
    register int _count;                                   \
    *outPtr++ = '[';                                       \
    for ( temp = symName, _count = symLen;                 \
          _count && *temp != '\r';                         \
          _count-- )                                       \
      *outPtr++ = *temp++;                                 \
    while ( _count-- )                                     \
      *outPtr++ = ' ';                                     \
    *outPtr++ = ']';                                       \
  }                                                        \
}
```

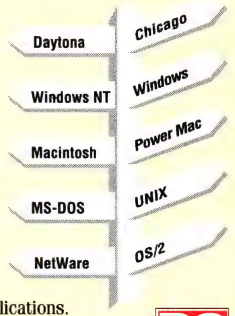
parseSymbols

```
/* parseSymbols() parses the symbol file.  Each element of name[] gets a pointer to a
symbol name.  Each element of value[] gets the value of a symbol.  Note that each
symbol name is terminated by '\r', not '\0'.  */

static unsigned short parseSymbols( char *buffer,
                                    char **name,
                                    ulong *value )
{
  register char *ptr;
  register char *temp;
  char **nextName;
  ulong *nextValue;
  ushort numSymbols;

  nextName = name;        /* where to save info for */
  nextValue = value;      /* the next symbol */
  numSymbols = 0;

  ptr = buffer;
  while ( *ptr )
  {
  /* The first 8 characters must be a hex number */
    XTOUL(ptr,*nextValue,temp)

  /* Skip whitespace between value and symbol */
    while ( *ptr++ == ' ' ) ;

  /* Save pointer to start of symbol */
    *nextName = ptr-1;

  /* Find start of next line */
    while ( *ptr++ != '\r' ) ;

    nextName++;
    nextValue++;
    numSymbols++;
  }

/ * Duplicate last symbol so the LOOKUP macro finds FFFFFFFF correctly.  */
  *nextValue = *(nextValue-1);

  return numSymbols;
}
```

convert

```
/* convert() converts 8 digit hex values in the input file.  It is also responsible for
writing the output bufffer to the output file.  */

static void convert(
            char *inputBuffer, char *outputBuffer,
            char **symbolName, ulong *symbolValue,
            ushort numSymbols, ushort symLength,
            size_t inputLength, FILE *outputFile )
{
  register char *inPtr, *outPtr;
  register char *temp;
  ulong addrValue, nextValue, *nextValuePointer;
  char *addrName, **nextName, *endPtr, *writeThreshold;

 /* inPtr and outPtr are both register variables.  The third address register variable is
temp.  */

  inPtr = inputBuffer;
  outPtr = outputBuffer;

 /* Stop when inPtr equals endPtr.  Flush the output buffer to disk when outPtr
exceeds writeThreshold.  */

  endPtr = inPtr + inputLength;
  writeThreshold = outputBuffer + OUTPUT_THRESHOLD;

 /* Add eight ASCII '0's to end of input file.  The search algorithm expects eight hex
digits after each \r, even at the end of the file.  */

  {
    temp = endPtr;
    *temp++ = '0'; *temp++ = '0';
    *temp++ = '0'; *temp++ = '0';
    *temp++ = '0'; *temp++ = '0';
    *temp++ = '0'; *temp++ = '0';
  }

/* Force lookup first time through.  */
  addrValue = 0xFFFFFFFF;
  nextValue = 0;      /* in case first address is FFFFFFFF */
  nextValuePointer = symbolValue-1;

/* Loop once per input line */
  while ( inPtr != endPtr )
  {

 /* Assume that the first eight characters of each line are all hex digits.  Convert this
value to [symbol+offset] form.  A line will usually have the same symbol as the
previous line, or sometimes the next higher symbol.  Thus, we check for these two
cases first and do a binary search only as a last resort.   */

    {
      register ulong address;
      XTOUL(inPtr,address,temp)
      if ( address < addrValue )
        goto lookup;   /* address going backward (unusual) */
      else if ( address >= nextValue )
      {
    /* Address doesn't match current symbol.  */
    /* Check if it matches the next symbol.  */
        addrValue = nextValue;
        nextValue = *(++nextValuePointer);
        if ( address < nextValue )
          addrName = *nextName++;  /* yes, it matches next */
        else
        {
     /* Doesn't match current or next symbol, so do a full binary search */
          register ulong index;
        lookup:
          LOOKUP(address,index,symbolValue,numSymbols,temp)
          addrName = symbolName[index];
```

WHETHER YOU'RE HEADED TO CHICAGO, DAYTONA, OR POINTS UNKNOWN, BASICSCRIPT HELPS YOU ARRIVE IN STYLE.



The BasicScript Toolkit makes it easy and economical to add an award-winning scripting language to your application.

- Easy-to-use APIs make it simple to integrate BasicScript into your application using C, C++, or other tools.
- Compatible with the syntax of Microsoft Visual Basic and Visual Basic for Applications (VBA).
- Controls OLE Automation objects in Macintosh, Windows, and Windows NT applications.
- Extensibility architecture allows you to add your own keywords to the BasicScript language.
- Available for Macintosh, Power Macintosh, Windows 3.1, Windows 95 "Chicago", Windows NT 3.5 "Daytona", MS-DOS, SunOS, Solaris 2.x, HP-UX, AIX, IRIX, UnixWare, SCO UNIX, Ultrix, OSF/1, Open VMS, NetWare, and OS/2.
- Runtime can be redistributed by your customer without royalties.
- Free technical support, comprehensive documentation, and numerous code samples keep your engineering costs to a minimum.
- Licensed by Symantec, Delrina, and more than three dozen other leading companies. Named *PC Magazine* Editors' Choice among cross-application macro languages.
- Flexible licensing terms mean you can afford the finest scripting language on the market today.

PC MAGAZINE EDITORS' CHOICE
February 8, 1994
The Norton Desktop 3.0 for Windows

BASICSCRIPT™

CALL TODAY FOR YOUR FREE EVALUATION COPY, 315 445.9000

Summit Software Company Fax: 315 445.9567 CIS: 71211,3504 Internet: info@summsoft.com WWW: http://www.summsoft.com

Solution Tools for HyperCard®, AppleScript®, OpenDoc® and More!

If you are a consultant, integrator, Mac manager, XCMD programmer or user, part developer, stack developer, or scripter, then you use Solution Tools.

If you don't already know about Heizer Software (we've been here since 1984) then you owe it to yourself to find out more.

Call, fax, or email us your address and we'll send you a FREE issue of our newsletter*(mention code H1102)*. We ***love*** HyperCard, so it's loaded with tools and newsbits about HyperCard. In the comming months, you'll see us offer more and more solution tools for AppleScript, OpenDoc, and more.

Here's a quick sampling of some of our more popular products:

CompileIt!™ – The HyperTalk® Compiler **$149** (30-0806)

WindowScript™ – The Interface Design Studio for HyperCard **$149** (30-0807)

PreFab Player™ – Script any application now **$95** (30-0808)

SCAuthor™ – Powerful tools for SuperCard® **$79** (30-0809)

HyperMarker™ – Quick and easy HTML editing **$25** (30-0810)

FileFlex™ – xbase database XCMDs that work wherever XCMDs do **$119** (30-0811)

Heizer Software

1941 Oak Park Blvd., #30 • P.O. Box 232019, Pleasant Hill, CA 94523
Orders: 800-888-7667 • International: 510-943-7667 • Fax: 510-943-6882
email: heizersw@aol.com • ftp://ftp.netcom.com/pub/he/heizersw

Over 100 AppleScript Scripts included on disk!
Scripting the Scriptable Finder
by Steve Michel
HEIZER SELECT SERIES

Scripting The Scriptable Finder by Steve Michel presents a lucid, complete discussion of how to use AppleScript to script the System 7.5 Scriptable Finder. Novice scripters and Mac Managers alike will find the information in this book invaluable.

The accompanying disk includes over 100 scripts along with popular scripting utilities such as Cron, OSAMenu, and FileTracker and a number of Scripting Additions. One of the best ways to learn scripting is to study existing scripts—the accompanying disk is a great resource for this purpose. **$49** (30-0812)

Steve Michel has worked as a freelance writer since 1987. He wrote the StackWEEK (later renamed "Script Manager") column for MacWEEK *from 1988 to 1994. He is the author of five Macintosh books and has written scores of software reviews for various publications.*

Shipping $3 USA/Canada; $7 International. Prices guaranteed through 8/95. Satisfaction guaranteed. CompileIt! & WindowScript are trademarks of Heizer Software. PreFab Player is a trademark of PreFab Software. SCAuthor is a trademark of Fourth World. HyperMarker is a trademark of Software Perspectives. FileFlex is a trademark of Component Software Corp. HyperCard, HyperTalk, AppleScript, OpenDoc, Mac, & Macintosh are registered trademarks of Apple Computer, Inc.

```
            nextName = symbolName + index + 1;
            nextValuePointer = symbolValue + index + 1;
            nextValue = *nextValuePointer;
            addrValue = *(nextValuePointer - 1);
          }
        }
        OUTPUT_ADDRESS(address-addrValue,addrName,
                       symLength,outPtr,temp)
    }

/* Scan the rest of the line for 8 digit hex numbers.  It is often possible to jump ahead
many characters when we find a non-hex digit.  We check seven characters ahead for a
hex digit, then six characters ahead, etc.  If we find a non-hex digit we know the
intervening characters can't possibly be an 8 digit number.  It turns out that if there is a \r
anywhere in the next eight characters, it will be the first non-hex character we encounter.
This happens because the first eight characters of a line are hex digits, and we are looking
ahead at most eight characters.  Thus, if we start beyond a \r, we will see hex digits all the
way back to the \r.  There can't be a second \r hiding earlier in the line since each line has
at least eight characters; another whole line wouldn't fit into the remaining characters.
When we jump to copy8..copy1, we only need to check for \r at copy1.    */

  /* Loop until end of line */
    while ( *inPtr != '\r' )
    {
      {
        temp = inPtr+7;
        if ( !ishex[ *temp-- ] ) goto copy8;
        if ( !ishex[ *temp-- ] ) goto copy7;
        if ( !ishex[ *temp-- ] ) goto copy6;
        if ( !ishex[ *temp-- ] ) goto copy5;
        if ( !ishex[ *temp-- ] ) goto copy4;
        if ( !ishex[ *temp-- ] ) goto copy3;
        if ( !ishex[ *temp-- ] ) goto copy2;
        if ( !ishex[ *temp ] ) goto copy1;
      }

      {
  /* Found 8 hex digits.  Convert to binary and ouput in "symbol+offset" form.    */

        register ulong value, index;

        XTOUL(inPtr,value,temp)
        LOOKUP(value,index,symbolValue,numSymbols,temp)
        OUTPUT_SYMBOL(value-symbolValue[index],
                      symbolName[index],outPtr,temp)
        continue;
      }

  /* Didn't have 8 hex digits.  Copy to output and check for \r.  */

      copy8:  *outPtr++ = *inPtr++;
      copy7:  *outPtr++ = *inPtr++;
      copy6:  *outPtr++ = *inPtr++;
      copy5:  *outPtr++ = *inPtr++;
      copy4:  *outPtr++ = *inPtr++;
      copy3:  *outPtr++ = *inPtr++;
      copy2:  *outPtr++ = *inPtr++;
      copy1:  if ( *inPtr != '\r' )
                *outPtr++ = *inPtr++;
              else break;  /* exit the inner while loop */
    }

  /* Now we're at the end of the line.  Copy the \r, */
  /* then flush the output buffer if we've run past the threshold.  */

    *outPtr++ = '\r';
    inPtr++;

    if ( outPtr >= writeThreshold )
    {
      fwrite(outputBuffer,1,outPtr-outputBuffer,outputFile);
      outPtr = outputBuffer;
    }
  }

/* We have reached the end of the input file.  */
/* Flush the rest of the output buffer.      */

  if ( outPtr != outputBuffer )
    fwrite(outputBuffer,1,outPtr-outputBuffer,outputFile);
}
```

By Neil Ticktin, Publisher

MacTech Launches eWorld Support

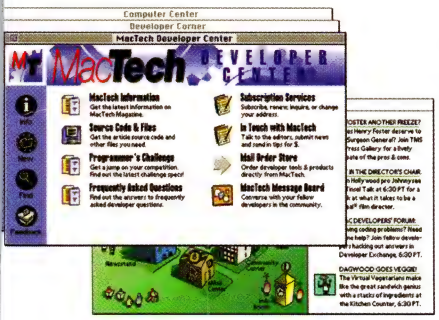
In 1994, Apple launched eWorld as their new online service. eWorld is now being shipped on every Macintosh going out the door. Ultimately, this service will completely replace AppleLink – the current tool used by Apple employees and developers alike. For those of you that don't know, AppleLink is an incredibly expensive service that General Electric Information Services has provided for Apple for many years now.

eWorld is based on technology licensed by Apple from America Online. And while you can see the similarities, Apple is providing much better service, less system problems, considerably shorter waits for (and better) customer service, a better set of publishing tools, and overall a much better experience than AOL as well as CompuServe.

Throughout 1995, Apple is going to be migrating people from AppleLink to eWorld. The first group to move is probably going to be Apple employees, but right up in front are developers. Don't worry, your AppleLink IDs have already been reserved on eWorld.

What's Available?

The MacTech area has several services – MacTech Information, Source and Other Files, Programmer's Challenge, Frequently Asked Questions (FAQs), Subscriber Services, In Touch with MacTech, Mail Order Store, and discussion areas.



MacTech Information
Get the latest information on MacTech Magazine.

Source Code & Files
Get the article source code and other files you need.

Programmer's Challenge
Get a jump on your competition. Find out the latest challenge specs!

Frequently Asked Questions
Find out the answers to frequently asked developer questions.

Subscription Services
Subscribe, renew, inquire, or change your address.

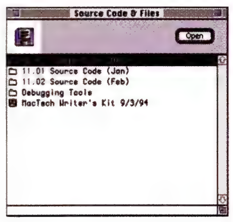
In Touch with MacTech
Talk to the editors, submit news and send in tips for $.

Mail Order Store
Order developer tools & products directly from MacTech.

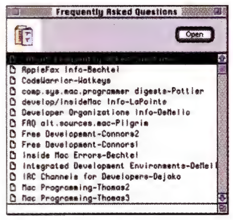
MacTech Message Board
Converse with your fellow developers in the community.

MacTech Information contains all of the latest information about the magazine. The Source and Other files will contain source code files that accompany articles, debugging tools, the latest versions of Sprocket, Apple's Universal Header Files and whatever else seems appropriate for a MacTech reader to have.

The Programmer's Challenge area will have the latest challenge questions – get a jump on your fellow developers by checking out the challenges early. The FAQ area is loosely based on MacTech's experience as the moderator of comp.sys.mac.programmer.info on the Internet. If you've got a question, check the FAQs first.



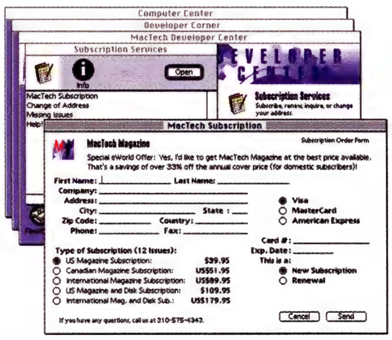
11.01 Source Code (Jan)
11.02 Source Code (Feb)
Debugging Tools
MacTech Writer's Kit 9/3/94



AppleFax Info-Bechtel
CodeWarrior-Hotkeys
comp.sys.mac.programmer digests-Pottier
develop/InsideMac Info-LaPointe
Developer Organizations Info-DeMello
FAQ alt.sources.mac-Pilgrim
Free Development-Connors2
Free Development-Connors1
Inside Mac Errors-Bechtel
Integrated Development Environments-DeMel
IRC Channels for Developers-Dejako
Mac Programming-Thomas2
Mac Programming-Thomas3

If you are a subscriber, or you want to subscribe, our Subscriber Services area has forms driven mail so that you can easily do what you need to. The same goes with our In Touch with MacTech area – write a letter to the editor, suggest an article, or submit a tip and get paid for it!

The Mail Order Store (MOS) is another great source for information. Let's say that you want to know about a particular programming tool or book. In the MOS, you can check out descriptions for all kinds of developer related products. If you'd like, there's also convenient ways to order, receive discounts and get customer service – all online!



MacTech Subscription
Change of Address
Missing Issues
Help!

Subscription Services
Subscribe, renew, inquire, or change your address.

MacTech Magazine — Subscription Order Form

Special eWorld Offer: Yes, I'd like to get MacTech Magazine at the best price available. That's a savings of over 33% off the annual cover price (for domestic subscribers)!

First Name: ______ Last Name: ______
Company: ______
Address: ______
City: ______ State : ______
Zip Code: ______ Country: ______
Phone: ______ Fax: ______

[✓ Visa] MasterCard American Express
Card #: ______
Exp. Date: ______

Type of Subscription (12 Issues):

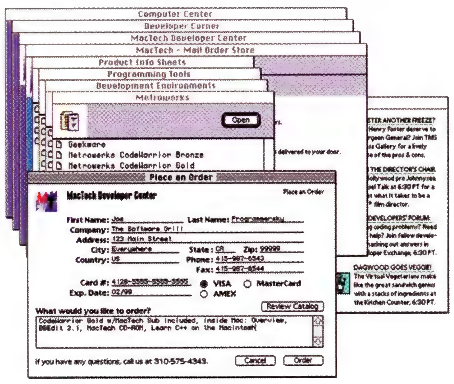
[✓ US Magazine Subscription:]	$39.95
Canadian Magazine Subscription:	US$51.95
International Magazine Subscription:	US$89.95
US Magazine and Disk Subscription:	$109.95
International Mag. and Disk Sub.:	US$179.95

This is a: [✓ New Subscription] Renewal

If you have any questions, call us at 310-575-4343.

How Do I Get There?

To explore the new MacTech Developer Center, go to the Town Square, then click on Computer Center, Developer Corner, and finally, MacTech Developer Center.



Geekware
Metrowerks CodeWarrior Bronze
Metrowerks CodeWarrior Gold

MacTech Developer Center — Place an Order

First Name: Joe Last Name: Programmersky
Company: The Software Grill
Address: 123 Main Street
City: Everywhere State : CA Zip: 99999
Country: US Phone: 415-987-6543
Fax: 415-987-6544
Card #: 4128-5555-5555-5555 [✓ VISA] MasterCard AMEX
Exp. Date: 02/99

What would you like to order?
CodeWarrior Gold w/MacTech Sub included, Inside Mac: Overview, BBEdit 2.1, MacTech CD-ROM, Learn C++ on the Macintosh

If you have any questions, call us at 310-575-4343.

Getting on eWorld

If you are interested in eWorld (as any Macintosh developer should be), you can get online by calling 800-775-4556 and asking for a starter kit.

INSIDE INFO

By Chris Espinosa, Apple Computer, MacTech Magazine Regular Contributor

I Was a Teenage Thought Policeman

It's a dirty job, but somebody's gotta do it!

Two months ago, MacTech published a letter from a reader talking about "good" and "bad" programming practices, calling the editors of this magazine the "Thought Police." The editors replied with something to the effect that somebody has to do it.

This issue crops up every few years, and people get really emotional about it. A lot of programmers naturally rebel at the thought of having Somebody Else tell them how they should write their code. The favorite historical Thought Police issue centers around the user interface, but error handling, C coding style, object design methodology have all seen conflict between some who would seek to impose standards and others who resist them.

I'm one of the people occasionally accused of starting all this in the Macintosh community. I wrote several of the early versions of the Macintosh Human Interface Guidelines, which got credited for creating the consistency among applications in the Mac system, but also roundly blamed for being the original Thought Police dictum of Macintosh political correctness.

The personal computer application software industry was really in its infancy when Mac development started in 1982, and developers really didn't know what elements made a software product a hit. Many apps were still "turn-key," meaning you started the computer directly into the app. Applications were free to take over the machine. Hard disks were rare, and floppies usually didn't hold more than two or three applications. So developers valued the individuality of their application, and in many cases considered a unique menu structure to be a major competitive advantage.

Apple proposed, pretty audaciously, that all applications should use a consistent menu command structure, and even more audaciously, use one that Apple designed. Even better, Apple built the interface software right into the ROM of the machine – it was in fact more difficult to use your own, familiar, already coded interface software than it was to knuckle under and do things Apple's way.

Reactions to this ranged from delight to umbrage. Some people conceptually bought the idea of consistency from application to application, and were enthralled by the amount of graphics, interface, and utility code already written for them, but others worried how they'd differentiate their products if all applications worked the same.

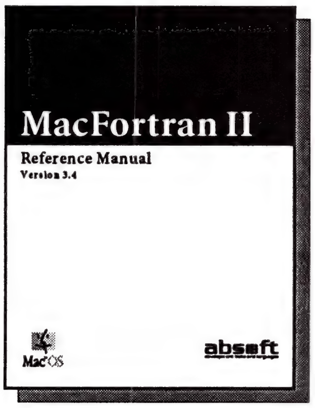
There was a pretty deep division of opinion over this. Many praised Apple for being daring and doing the Right Thing, and defended the guidelines. Others resented the fact that it was so hard to not do things the Macintosh way, and condemned the platform for being proscriptive. There was a little bit of a war between these forces, and most people assumed that Apple was on the side of the consistency zealots.

And I'm certain there were influential people at Apple who did some righteous bashing of inconsistent applications. I remember an early database, MacLion, which was a bad port of a DOS application, right down to the 24-by-80 monospaced scrolling text window. Boy, it was ugly. It eventually lost in the marketplace. Apple also spent a lot of time working with major DOS application vendors to get them to "get it" about the graphic user interface. Lotus received a lot of personal attention from Apple for their Jazz product, and later 1-2-3 for Mac.

But the folklore that has come down through the years is that Apple defended the purity of the interface by punishing the developers who built applications that broke the rules. And that's just not true. The rules were vague; they were revised several times over the first five years; we broke the rules ourselves (starting early, with MacPaint); and to tell you the

NEW Absoft FORTRAN 77 _Updates_

New releases of the world's leading FORTRAN for Macintosh



MacFortran II v3.4

Optimized for Mac II/Quadra CPUs

Each new release includes:

- *Fully validated ANSI X3.9 native compiler*
- *CPU-specific optimizations for fastest execution*
- *Compiles up to 5X faster*
- *New Absoft CreateMake utility*
- *Source-level symbolic debugger*
- *Two complete graphics libraries*
- *MRWE application framework*
- *Full System 7.5 compatibility*
- *Latest version of Apple's MPW*

Absoft F77 SDK v4.1

Optimized for Power Macintosh

Contact Absoft for details and upgrade pricing. Academic and volume discounts available.

Tel: (810) 853-0050 • Fax: (810) 853-0108 • AppleLink: absoft • Internet: fortran@absoft.com

---

truth, we were so desperate for software that we even put that ugly, DOSish MacLion on our poster of the first 100 apps.

The truth is that the punishment for inconsistency came from the Mac community itself. Magazine reviewers and pundits were the first to appreciate the consistency and simplicity of Mac applications, especially in contrast with the growing mess in the DOS world. Influential users and purchasers followed suit. Programs with inconsistent interfaces did suffer; but they suffered at the hands of the marketplace, not of a dictatorial Apple.

By 1991, Apple's efforts around human interface were pretty much limited to advice, pronouncements, and leading by example (such as in System 7's outline-expansion arrows and Drag and Drop). Users and editors still screamed at developers, and developers at each other, oddly turning the original fear on its head: developers claimed their application was superior because it was more conformant to the standard, not unique and different. The interface grew and changed, with some styles becoming popular (like windoids) and others just not making it (such as the use of Microsoft-style boxes to group elements in a dialog box). The idea that there were Thought Police, though, was so thoroughly ingrained in the Mac community that the thought police didn't need to exist anymore: developers just did the right thing the majority of the time, but still took risks once in a while to push the interface forward.

There's always a balance between freedom and responsibility. Programmers want to be free, but in order for the whole community to be successful, programmers have to be responsible, and hold up their end. Keeping the interface consistent, reducing bugs by doing correct error handling, avoiding inappropriate system hacks, and not being predatory of competitive applications are some pretty important responsibilities. Doing these well improves the quality of life for all Mac users, and supports the platform sales, creating a larger opportunity for software authors.

Scott Boyd is perfectly right: somebody has to be the thought police. The beauty of the last ten years of Macintosh development is that *everybody* is the Thought Police. You've got a lot of people looking over your shoulder to keep you straight, but there's no actual bogeyman there to beat you up if you get creative.

C++ GRAPHICS CLASS

By Eric Rosé, cp3a@andrew.cmu.edu

Homerolled Hierarchies

C++ objects to manage and draw trees dynamically

Introduction

In the past thirty years, a considerable amount of research has been done on the automatic drawing of graphs (that's the node-and-arc kind – not the Excel kind). Of special interest to interface designers are algorithms which can be used to display a special kind of graph: the hierarchical tree. Trees are a natural way to represent information which can be hierarchically subdivided, such as: organizational charts, design spaces, directory structures, common data structures like binary search trees, B-trees, and AVL-trees. In the last ten years or so there have been many papers which discuss algorithms for aesthetically laying out hierarchical trees (see references), though few of them are intended to do so dynamically. This article discusses two strategies for drawing dynamically changing hierarchical trees, and provides a set of five C++ classes which use these strategies to draw trees while taking into account several customizable display options.

Since the code is rather sizable, I will only present the class interfaces and important parts of the method implementations. The source code disk includes not only the full source code for these classes, but the code for an application which I wrote to test all of the features of both tree algorithms.

Algorithms

Both of the following algorithms assume that each node in a tree consists of a pointer back to its parent, a linked list of child nodes, and a block of data which it can compute the size of. This provides a structure which is very easy to recursively traverse.

Algorithm ER

The first algorithm is a recursive routine which I devised in response to a challenge in a data structures class. It positions nodes by recursively calculating rectangles which enclose successively larger subtrees and justifying the parent of the subtree within that rectangle.

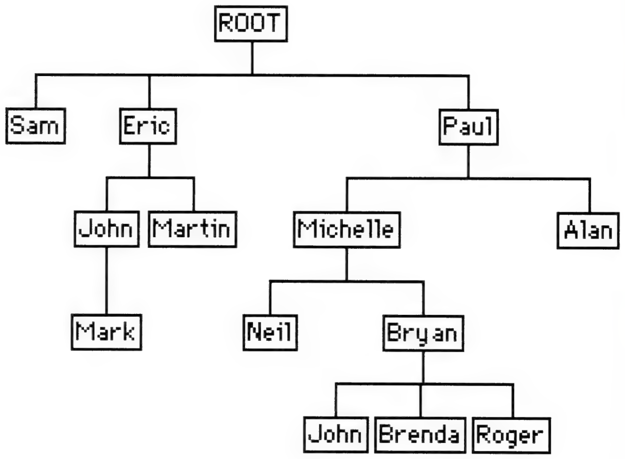


Figure 1. Decomposition of a tree by algorithm ER

Eric Rosé – A recent escapee from a Masters program in Electrical and Computer Engineering, Eric writes software for medical treatment systems by day. By night Eric does Macintosh consulting and pursues his interests in software development environments, neat interface hacks, and on-line substitutes for the books which are overflowing his apartment. You can reach him for comment at cp3a@andrew.cmu.edu.

ER takes as parameters a node to place, and its desired top left corner. If the node does not have any children, the node's top left corner is assigned to be the passed-in coordinate. If the node does have children, ER calls itself for each child, accumulating the width of all previous subtrees and adding it to the passed-in coordinate at each call. This insures that every node will be to the right of the subtrees which come before it. When all the children have been considered, the algorithm centers the parent over the children and then exits. A pseudo-code version of ER is shown below.

```
1 ER (Point topLeft, node NodeToPlace) {
2   Point  TL = topLeft

3   if nodeToPlace has children
    {
4     TL.v += child/parent margin
5     for each child of NodeToPlace
      {
6       ER (TL, child)
7       TL.h += width of child's subtree + sibling margin
      }
8     justify NodeToPlace over its children
    }
9   else
10    NodeToPlace's top left corner = topLeft
  }
```

Note that if we simply change lines 4 and 7 as shown below, we get a tree which is oriented horizontally rather than vertically.

```
4     TL.h += child/parent margin
7       TL.v += height of child's subtree + sibling margin
```

The trees drawn by algorithm ER are aesthetically pleasing, and good for displaying trees with fairly evenly sized subtrees. A drawback is that it does not make any attempts to reduce white space by packing nodes more closely together. For example, opportunistic compression of the subtree whose root is "Paul" would produce the subtree shown in Figure 2, which is approximately 75% as wide.

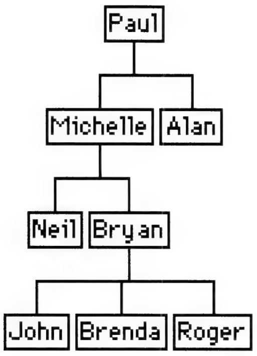


Figure 2. Opportunistic Compression of the sample tree

Algorithm SG

The second algorithm, also recursion-based, is used in a graphics visualization tool called SAGE, developed at Carnegie Mellon's robotics institute. It works by making position estimates which place each node as close as possible to its siblings, and then calculating their real positions as it comes back out of the recursion. A pseudo-code version of SG is shown below, followed by an English description.

```
1 SG (Node NodeToPlace, short level, short levelOffset) {
2   if NodeToPlace's estTopLeft.h is greater than Contour[level]
3     set estTopLeft.h to Contour[level];
4   if NodeToPlace has children
    {
5     for each child of NodeToPlace
      {
6       estimate the child's topleft position
7       SG (child, level+1, levelOffset);
      }
    }
8   Justify the node and fix its position
9   Contour[level] = the node's rightmost position
  }
```

Again, note that changing lines 2, 3, and 9 as shown below will effectively change the orientation of the tree to horizontal instead of vertical.

```
2   if NodeToPlace's estTopLeft.v is greater than Contour[level]
3     set estTopLeft.v to Contour[level];
9   Contour[level] = the node's bottommost position
```

Before SG begins, we create an array called Contour with an entry for each level of the tree. This array is used to store the rightmost position where a node may be placed; all entries are initialized to zero. SG takes as parameters a node to place, its level in the tree, and the vertical distance between it and its parent. If the node has children, then for each child it first estimates the position of the child's top left corner and then calls SG on it. Estimates are made in the following way: it places the first child by taking the cumulative width of all the children, dividing that width in half, and subtracting it from its estimated center (i.e., its estimated top-left position plus half its width). Each subsequent child is placed by adding the between-node margin to the value in the Contour array for that level of the tree.

"... it would be nice to be able to change the orientation of a tree from top-down to left-right by tweaking a parameter."

When SG first considers a node (i.e., on its way into the recursion) it checks the node's estimated top-left position against the value in the Contour array. If it is smaller than the value in the array, it is replaced with the value in the array plus the between-node margin. This insures that each node is placed as close as possible to its left sibling. When SG considers the same node on its way out of the recursion, it revises its estimate of the node's position by centering it over its children. If it has no children, the estimate is not revised at all. Having fixed the top left position, SG updates the value of the Contour array to correspond to the node's right border. In this way, SG insures that the node's right sibling will be placed correctly.

Figure 3 provides a visual walk-through of the way in which SG would operate on a simple tree. (Gray frames indicate nodes whose positions are estimations. Black frames indicate nodes whose positions have been fixed.)

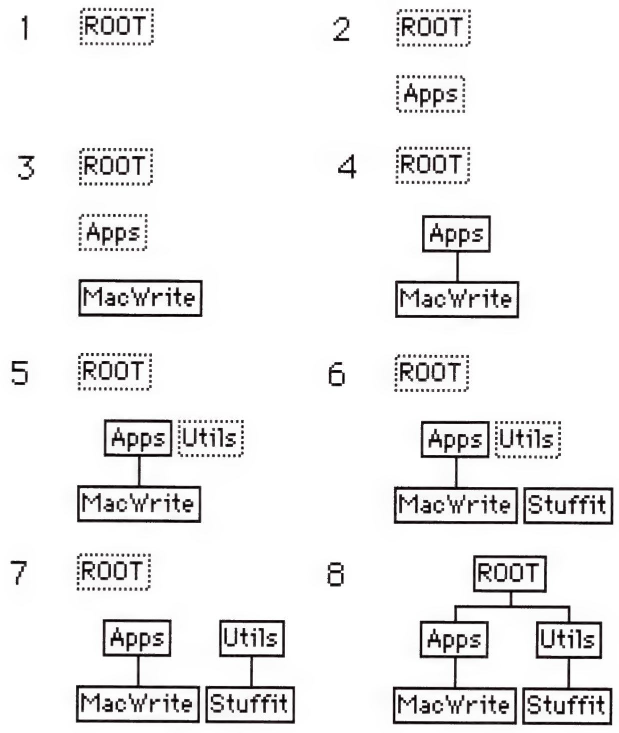


Figure 3. Visual trace of algorithm SG

The Creeping Feature Creature

Many of the algorithms which are discussed in the literature have been developed to address specific instances of the tree drawing problem; ie. top-down binary trees, trees with fixed-size nodes, etc. When I set about designing the tree-drawing classes, I decided that it would be useful to build in a lot of flexibility. For example, it would be nice to be able to change the orientation of a tree from top-down to left-right by just tweaking a parameter. I finally decided on seven characteristics which should be dynamically modifiable:

1. size, shape, and data content of nodes in the tree
2. number of children per node
3. justification (left, center, right) of nodes over their children.
4. minimum distance between a node and its siblings
5. minimum distance between a node and its children
6. orientation (top-down, bottom-up, left-to-right, right-to-left) of the tree
7. how lines are drawn between nodes and their children (right-angle, point-to-point)

Since the point of the exercise is to be able to dynamically modify the tree, here's the list of operations we want to support:

1. Add a child
2. Insert a child
3. Insert a parent
4. Delete a child and promote its children
5. Delete a child and all of its children
6. Change the content (and so possibly the size and shape) of a node

As if that wasn't enough, we will also add the restriction that the only nodes which should be redrawn when the above operations are performed are the ones whose contents or positions change. There are two reasons for this. The first is to reduce unsightly flicker. For those who respond to this reason by saying "use an offscreen GWorld, dummy", the second reason is to save time spent doing redraw. While offscreen GWorlds do reduce flicker, redrawing every node on every operation causes unpleasant delays if you have a large number of moderately complex nodes (trust me – I fought this problem all summer long.)

Anybody appalled yet? Relax – everything except the last restriction is pretty straightforward (though the switch statements do get a bit daunting in places!).

OOP(s)!

"The time has come", the Walrus said, "to talk of implementation details." In other words, given the algorithms we are using and the features we want, how can we partition everything into classes? One intuitive approach (and, in fact, the one I use) is to treat both the tree and the nodes within the tree as independent objects. Since nodes are objects, their data content (and consequently their size and shape) can be controlled through subclassing. This provides the first feature on our list. The second feature is easily provided by keeping children in an expandable linked-list. The last five features really apply to each tree as a whole since it could be kind of jarring to have orientation, justification, etc. change from node to node. We will, therefore, store information for the last five features in the tree object.

The drawing algorithms we use also map naturally onto the object model proposed above, since they both work by considering successively larger subtrees, rather than dealing with the tree as a whole. This should (and does) allow us to shift the burden of calculation onto the nodes themselves.

CPPTree and CPPTreeNode

Since trees are such massively useful data structures, I decided to start by building two classes which would let me build trees in memory without having to keep track of any display-specific information. The interfaces for the tree class is shown below.

```
class CPPTree : public CPPGossipMonger {
public:
        CPPTree (void);
        ~CPPTree (void);
  virtual  Boolean  Member (char *className);
  virtual  char *ClassName (void);
```

Let's Table the discussion.

Take data from multiple sources and create an **unlimited** number of rows and columns made up of text, movies, pictures, ICONs, SICNs and other types of information.

A**ppleScript** support provides integratation with other applications. Full set of object resolvers provided.

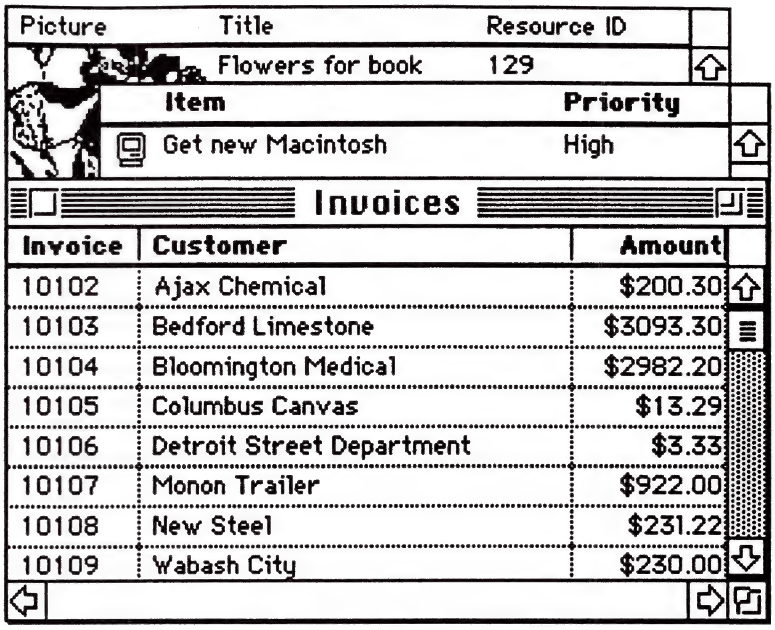
Build powerful tables in your C/C++ environments by replacing the list manager with TableIt!. Our drawing procedures make LDEF's a thing of the past in under 5 minutes.

Leap into the future with support for **OpenDoc™**, the exciting new compound document architecture. Combine TableIt! parts with other commercial and custom parts to provide exciting and powerful solutions.

Enlarge the amount of data you can work with in your application by combining TableIt! with NeoAcess™ - the powerful cross-platform object-database. No more RAM limits!

I**ntegrate** TableIt! with your current applica tion environments - 68K and PowerPC versions are available for 4D, HyperCard, SuperCard as well as most C/C++ compilers. *Certified* PowerPC native for CodeWarrior.

Test drive a version on a 30 money back guarantee.



Picture	Title	Resource ID
	Flowers for book	129

Item	Priority
Get new Macintosh	High

Invoices

Invoice	Customer	Amount
10102	Ajax Chemical	$200.30
10103	Bedford Limestone	$3093.30
10104	Bloomington Medical	$2982.20
10105	Columbus Canvas	$13.29
10106	Detroit Street Department	$3.33
10107	Monon Trailer	$922.00
10108	New Steel	$231.22
10109	Wabash City	$230.00

TableIt! Features:

• Unlimited number of rows and columns - go past the typical 32K limitation!
• Support for core, database, table and text suites. Playable, Recordable and Attachable.
• Over 100 properties that provide customized look and feel.
• Drag and Drop Support.
• In-cell editing
• DR/2 for OpenDoc available now!

Pricing:

$150 retail for stand-alone, HyperCard, SuperCard or 4D. Developers Licenses and OpenDoc part licenses available. No Royalties!

Graphical Business Interfaces, Inc.
Voice: 219-253-8623 Fax: 219-253-7158
E-Mail: info@gbi.com
http://www.gbi.com

TableIt!™ Bringing your data into view.™

```
       void     SetTopMostNode (CPPTreeNode *theNode);
  inline CPPTreeNode*GetTopMostNode (void)
                                {return this->topNode;}
  virtual  void ReceiveMessage (CPPGossipMonger *toldBy,
                                short reason, void* info);
       short  Depth (void);

  CPPTreeNode*topNode;
};
```

CPPTree is descended from a class called GossipMonger, which is functionally identical to the CCollaborator object in the TCL. For those of you unfamiliar with CCollaborator, it lets you form dependencies between objects so that messages can be automatically propagated between them. Messages are sent using a method called `BroadcastMessage`. The `ReceiveMessage` method in the target object is responsible for catching these messages and either delegating or dealing with them appropriately. Note that if you have a pointer to CPPTree you don't have to form a specific dependency; you can just call `ReceiveMessage` directly.

The only data which the tree class stores is `topNode` – a pointer to a `CPPTreeNode` object which is the root node of the tree. `GetTopMostNode` and `SetTopMostNode` let the user of the class retrieve or change the root node, and the `Depth` function returns the number of levels in the tree.

The interface for the tree node class is more elaborate, since it has to deal with all of the operations we listed earlier. Its interface is shown below.

```
class CPPTreeNode : public CPPObjectList {
public:
  CPPTree      *Root;
  CPPTreeNode  *Parent;

      CPPTreeNode (CPPObject *NodeData,
              CPPTree *BelongsTo,
              Boolean becomeOwner);
      ~CPPTreeNode (void);
  virtual  char     *ClassName (void);
  virtual  Boolean  Member (char *className);
  virtual CPPObject *Clone(void);

      void TellParent (short Message, void *data);

      CPPTreeNode*NthChild (long whichChild);

      void AddNode (CPPTreeNode *NewNode);
      void InsertNode (CPPTreeNode *NewNode, long insertWhere);

      Boolean InsertParent (CPPTreeNode *NewNode);
      void RemoveChild (CPPTreeNode *theChild);
      void SwitchParents (CPPTreeNode *OldParent,
                          CPPTreeNode *NewParent);

      short  FamilyDepth (short currentDepth);

      CPPObject  *GetNodeData (void);
      void   SetNodeData (CPPObject *NewData,
                          Boolean becomeOwner);

  virtual  void ReceiveMessage (CPPGossipMonger *toldBy,
                       short reason, void* info);
  virtual  void GetFamilyBounds (Rect *bounds);

  static Boolean  DeleteNode (CPPTreeNode *theNode);
  static void   DeleteFamily (CPPTreeNode *theNode);
  static void ClearFamily (CPPTreeNode *theNode);
  static long FamilySize (CPPTreeNode *theNode);

protected:
  CPPObject  *nodeData;
  Boolean    ownsData;
};
```

CPPTreeNode is descended from CPPObjectList, which lets you store an ordered list of C++ objects. The contents of the list, in this case, will be the children of the node. CPPTreeNode holds four pieces of data. The first is a pointer to its own parent. This is kept so that it can pass messages up to its parent if either it or its children change. The second is a pointer to the CPPTree to which the node belongs. This effectively lets the node inform the tree directly if it changes (kind of like going over your boss' head to the CEO). Lastly, CPPTreeNode keeps a pointer to an object which contains node-specific data, and a flag indicating whether or not the node owns (and therefore is responsible for disposing of) that object.

"Only the nodes which change are redrawn."

Why, you may ask, did I choose to have each node track a separate object instead of letting them track their own specific data and information about drawing it? Primarily to preserve modularity and provide for object reuse. By way of example, if I wanted a node which displays pictures, I could put all the picture-specific retrieving, sizing, and drawing functions inside the node. This would contaminate the object by mixing information about two completely different tasks – fiddling with pictures and being a node in a tree. If I instead create an object which knows all about pictures and let the node manage that object I win in two ways: 1) I can preserve the modularity of both the node and the picture objects, and 2) I can use the picture object anywhere else in my program without dragging along all the baggage needed to maintain tree information (and vice versa).

Enough about the data; let's quickly review the methods which CPPTreeNode offers. The first significant method is `TellParent` which lets the node pass a message to its parent (if any). This method is used heavily in the subclass of CPPTreeNode which handles the display of the node. `NthChild` is used to return a pointer to one of the children of the current node. `AddNode` and `InsertNode` let you add a child either to the end or middle of the node's list of children. `InsertParent` lets you insert a node between your parent and yourself (I know it sounds odd, but it is nice to have around sometimes). `RemoveChild` detaches the specified child and its subtree from the parent without deleting the subtree. `SwitchParents` lets you move a subtree from one parent to another in one step. `FamilyDepth` gives the number of levels in the node's subtree; calling `FamilyDepth`

When you can't afford to slip, and the product just has to ship...

call

THE MAC GROUP

intense, focused debugging
aimed at helping you ship

800 SyncWait
796-2924

24 hours – we make housecalls

---

on the top node of the tree will give you the same result as `CPPTree::Depth`. `GetNodeData` and `SetNodeData` let you retrieve and change the data held by the node. `GetFamilyBounds` is a virtual method which doesn't do anything in CPPTreeNode.

Finally there are four class methods – `DeleteNode`, `DeleteFamily`, `ClearFamily`, and `FamilySize`. `DeleteNode` deletes the passed-in node, but adds all of its children to the node's parent. `DeleteFamily` deletes the passed-in node and its entire subtree. These two methods should be called to get rid of structures which are being displayed on the screen. `ClearFamily` performs the same function as `DeleteFamily`, but assumes that you are deleting a structure which is not being displayed. The last routine – `FamilySize` – simply returns the number of nodes in the passed-in subtree.

Two more useful functions defined in CPPVisualTreeNode.h are `ApplyToFamily` and `BApplyToFamily` – shown below:

```
void ApplyToFamily (CPPTreeNode *familyHead, ApplyProc theProc,
                    long param);
Boolean BApplyToFamily (CPPTreeNode *familyHead, BApplyProc theProc,
                    long param);
```

These routines let you do a postorder traversal of the subtree whose head is passed in `familyHead`, applying a routine which accepts a long parameter to each node it encounters. Using these routines saves you the trouble of having to write code to traverse subtrees yourself. For example, the `FamilySize` routine is implemented in the following way:

```
long CPPTreeNode::FamilySize (CPPTreeNode *theNode)
{
  long count = 0;
  if (theNode)
    ApplyToFamily (theNode, CountChildren, (long)(&count));
  return count;
}

void CountChildren (CPPTreeNode *theNode, long param)
{
  (*((long *)param))++;
}
```

CPPVisualTree

CPPVisualTree is the class which deals with displaying trees on the screen. To save space here, its interface – considerably more complex than that of its superclass – is shown at the end of the article. To support the last five features from our feature list, it defines the following variables:

```
orientStyle orientation;
justStyle   justification;
short       branchLength;
short       nodeMargin;
joinTypes   whichJoin;
```

`Orientation` determines the orientation in which the tree is drawn. It can have one of four values: kTopDown, kBottomUp, kLeft2Right, and kRight2Left. `Justification` determines how nodes are placed over their children. It can have one of five values: kJustCenter, kJustRight, kJustLeft, kJustTop, and kJustBottom. kJustTop and kJustRight are equivalent, as are kJustBottom and kJustLeft; the different name is included for readability since its hard to imagine what right justification means in a horizontally oriented tree. `BranchLength` determines the minimum distance between a child and its parent, and `nodeMargin` determines the minimum distance between sibling nodes.

`WhichJoin` determines how lines are drawn between nodes. It has three predefined settings – kRightAngle (which is the style used in the examples), kPointToPoint (which simply draws lines directly from a parent to its children), and kNone (which results in no lines being drawn at all). CPPVisualTree's protected methods `DrawPoint2Point`, and `DrawRightAngle` handle drawing the first two kinds of joins for you. You can add other kinds of joins by either adding constants and methods to CPPVisualTree or subclassing the DrawJoin method in the visual tree node class. More on that later.

CPPVisualTree has one accessor and one setter method for each of the variables defined above. The setter methods (one of which is shown below) all have a similar structure – differing predictably in lines 1 and 5:

```
1 void CPPVisualTree::SetBranchLength (short newLength)
  {
2   if (this->branchLength != newLength)
      {
3     this->Prepare(this);
4     CPPTree::BroadcastMessage (kEraseFamily, (void *)TRUE);
5     this->branchLength = newLength;
6     CPPTree::BroadcastMessage (kResizeFamily, NULL);
7     CPPTree::BroadcastMessage (kDrawFamily, (void *)TRUE);
8     this->Restore(this);
      }
  }
```

Line 2 tests to see if the new setting and the old are, indeed, different. If so, it calls `Prepare` which sets up the grafport so that the tree can draw properly. It then uses `BroadcastMessage` to erase the tree, changes the old setting, then calls it again to tell the tree to resize and draw itself before restoring the drawing environment.

A short note on `Prepare` and `Restore`. The object which is passed to them is used as a sort of key so that `Prepare` will not actually set up the drawing environment more than once when a tree is drawn. Whenever a node is drawn it first calls `Prepare`, draws itself, then calls `Restore`. This is because we cannot guarantee that the tree's window is the front window when a node is added or deleted. Whenever `Prepare` is called it first checks the class variable `isPrepared` to see if the grafport is already set up. If it isn't, it saves a reference to the object in the variable `preparedBy`, saves the current grafport in the variable SavePort, and sets the current port to the window where the tree lives. When `Restore` is called, it checks the passed-in object against the one in `preparedBy`, and, if they are identical, restores the drawing environment in `SavePort`. Make sense? If not, don't worry – we'll see more about how this mechanism is used later. Two protected virtual methods – `UserSpecificPrepare` and `UserSpecificRestore` let your tree class do any special setup which might be necessary for the tree's grafport.

Five methods are provided for dealing with the display of the tree. The first, `ForceRedraw`, causes the entire tree to draw itself (after optionally erasing itself). The second, `ForceResize`, causes all of the nodes to recalculate their sizes and positions. You can ask this method to explicitly erase the whole tree and then draw it afterwards. The third method, `ForceMove`, causes the entire tree to move to a new location. When a CPPVisualTree object is created, you pass it a point to use as its top left corner; `ForceMove` changes this top left corner and causes all of the nodes to reposition themselves accordingly. The fourth method, `DrawAllJoins`, causes all of the joins in the tree to be drawn or erased, depending on a boolean which you pass in. The last method is used to draw the tree in response to an update event, and is called (rather predictably) `Draw`.

Since CPPVisualTree is intended for use with dynamically modifiable trees, two methods – DoTreeClick and DoCommand – are provided for responding to user input. Passing an event record to DoTreeClick will handle all click/drag-selection of nodes within the tree. DoCommand is provided to let you respond to commands like Cut, Copy, Paste, Clear, etc. although currently only Clear is implemented.

CPPVisualTreeNode

CPPVisualTreeNode is the base class for all nodes which appear in visual trees. Rather than have it be merely a virtual base class, I set it up to implement algorithm ER which, despite its shortcomings, is perfectly adequate as a default positioning routine. Its interface is also quite lengthy, and is provided for your perusal at the end of this article. Following is a discussion of its key features.

A node and its subtree can be described by four (yes, four!) overlapping rectangles, each of which is a class variable. A diagram of these rectangles is shown in Figure 4.

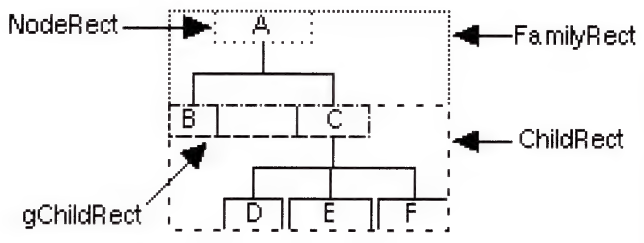


Figure 4. Rectangles used to describe A's subtree

`NodeRect` is the smallest rectangle which encompasses the current node. `gChildRect` is the smallest rectangle which encompasses all of its immediate children. `FamilyRect` is the smallest rectangle which encompasses the entire subtree of which A is the head, and `ChildRect` is the smallest rectangle which encompasses the `FamilyRect`'s of A's immediate children. If you look back at Figure 1, you can see that the rectangles which ER uses to position subtrees correspond to the `FamilyRect` frame in Figure 4. No coincidence! These four rectangles are defined by a method called `DoCalcFamilyBounds` which is actually the heart of the implementation of algorithm ER. I'm not going to go into a discussion of this method, except to point out the parts that implement the "draw only when necessary" feature, since the C code is a fairly straightforward implementation of the pseudo-code.

CPPVisualTreeNode also stores three points – `nodeSize`, `childSize`, and `familySize`, which track the horizontal and vertical extents of the rectangles described above. These extents are filled in by a method called `CalcFamilySize` which is always called before `CalcFamilyBounds`. The reason for this is that all of the extents can be computed without having to know exactly where the frames go. We'll get into more depth on how `CalcFamilyBounds` and `CalcFamilySize` work together in a minute.

But first, let's quickly go over the routines which the node uses to draw itself and its subtrees. CPPVisualTreeNode has five routines corresponding to the accessor routines from CPPVisualTree (`GetOrientation`, etc.) which simply call the tree's accessor routines through the `Root` field of `CPPTreeNode`. These are used to get the orientation/justification information which ER uses to position everything. There are two routines – `DrawJoin` and `EraseJoin` which draw and erase the line between a node and its children. Currently they both call CPPVisualTree's `DrawDefaultJoin` method, but they can be overridden to provide any kind of custom join you want. Associated with `DrawJoin` and `EraseJoin` is a routine called `GetAnchorPoints` which returns the points on the left, right, top, and bottom of the node where joins are to be drawn to and from. Currently these points are calculated as the centers of each side of the node.

Two more routines – `DrawNode` and `EraseNode` – set up the drawing environment for the tree, then draw or erase either the single node or its entire subtree. A virtual routine called `DrawNodeData` (which you have to provide the guts for) does the actual drawing of the special data held by the node. Another virtual routine which you have to provide the guts for is `CalcNodeSize`, which should figure out how big the node's data is and store it's extent in the `nodeSize` variable.

We are now ready to talk about how the "redraw when necessary" feature is implemented for the operations we want to perform on our tree. You will notice three variables – `needsResize`, `needsMove`, and `needsDraw` – and one method – `CanDraw` – in CPPVisualTreeNode's interface. These flags are used to keep nodes which do not need to be resized, moved, or drawn from passing through `CalcNodeSize`, `CalcFamilyBounds`, and `DrawNode`. This lets us be even more efficient, since it reduces unnecessary size and boundary calculations as well as unnecessary drawing. Whenever a node passes through ER, `needsResize` and `needsMove` are set to FALSE, and needsDraw is set to TRUE only if the node has actually moved. From this point on, no node is moved or resized unless one of the first two flags is explicitly set to TRUE.

But where *do* they get set to TRUE? Glad you asked. They get set in CPPVisualTreeNode's `ReceiveMessage` method, shown below.

```
void CPPVisualTreeNode::ReceiveMessage (CPPGossipMonger *toldBy,
                    short reason, void* info)
{
  Point  oldTopLeft = {this->familyRect.top,
                       this->familyRect.left};
  Rect oldFamilyRect = this->familyRect;
  Point  OldFamilySize = this->familySize,
         OldChildSize = this->childSize,
         OldNodeSize = this->nodeSize;
  Boolean  childChanged, familyChanged;
  RgnHandle  Rgn1, Rgn2;

  static short  timesCalled = 0;  // tracks recursion

  switch (reason) {
    case kEraseFamily :
      EraseNode ((Boolean)info);
      break;

    case kDrawFamily  :
      DrawNode ((Boolean)info, TRUE);
      break;

    case kResizeFamily  :
      ResizeFamily (TRUE);
      if (this->Root)
        ((CPPVisualTree *)this->Root)->AdjustTree();
      break;

    case kMoveFamily  :
      MoveFamily ((Point *)info);
      if (this->Root)
        ((CPPVisualTree *)this->Root)->AdjustTree();
      break;

    case kPrepareRemove :
    case kPrepareDelete :
      ((CPPVisualTreeNode *)info)->EraseNode(TRUE);
      ApplyToFamily((CPPVisualTreeNode *)info,
                    SetNotYetPlaced, TRUE);
      break;

    case kChildRemoved  :
    case kChildDeleted  :
    case kNodeAdded :
    case kNodeChangedData :
      timesCalled++;
      if (timesCalled == 1)
        if (this->Root)
          ((CPPVisualTree *)this->Root)->
                          Prepare((CPPObject *)1313L);

      this->needsResize = TRUE;
      this->needsMove = TRUE;
      CalcFamilySize();
      CalcFamilyBounds (oldTopLeft);
      if (info)
        this->needsDraw = TRUE;

      if (this->Parent && !EqualRect(&oldFamilyRect,
                            &this->familyRect))
        TellParent (reason, NULL);

      if (this->needsDraw)
        {
        this->DrawNode(TRUE, FALSE);
        if (this->Root)
            ((CPPVisualTree *)this->Root)->
                              DrawAllJoins(TRUE);
        if ((OldFamilySize.h > this->familySize.h) ||
           (OldFamilySize.v > this->familySize.v))
          {
          Rgn1 = NewRgn();
          Rgn2 = NewRgn();
          RectRgn(Rgn1, &oldFamilyRect);
          RectRgn(Rgn2, &this->familyRect);
          XorRgn (Rgn1, Rgn2, Rgn1);
          EraseRgn (Rgn1);
          DisposeRgn(Rgn1);
          DisposeRgn(Rgn2);
          }
        if (this->Root)
          ((CPPVisualTree *)this->Root)->AdjustTree();
        }

      timesCalled--;
      if (timesCalled == 0)
        if (this->Root)
        ((CPPVisualTree *)this->Root)->
                          Restore((CPPObject *)1313L);
      break;

    default:
        CPPTreeNode::ReceiveMessage (toldBy, reason, info);
        break;
    }
}
```

Remember when we talked about ReceiveMessage back in the discussion of CPPTree? Whenever we call a method in CPPTreeNode corresponding to one of the six operations (InsertNode, RemoveChild, etc.) it uses ReceiveMessage to tell the visual display to update. The implementation of InsertNode is shown below as an example.

```
void CPPTreeNode::InsertNode (CPPTreeNode *NewNode,
                                    long insertWhere) {
  if (NewNode) {
    NewNode->Parent = this;
    ApplyToFamily (NewNode, SetRoot, (long)this->Root);
    InsertItem (NewNode, insertWhere);
    this->ReceiveMessage (this, kNodeAdded, (void *)NewNode);
  }
}
```

The sixth case in ReceiveMessage is the code segment we are really interested in. This is the heart of the "redraw when necessary" technique. You will notice that it gets called whenever a node is inserted, deleted, removed, or when the node's contents change. When you consider that operations 3 and 4 (insert parent, delete & promote children) use operations 1, 2, and 4, you can see that this bit of code gets called eventually for every one of the six operations we support on trees.

So how does this piece of code work? Well, let's consider the case where we want to add a child to node "Two" in the tree shown below.

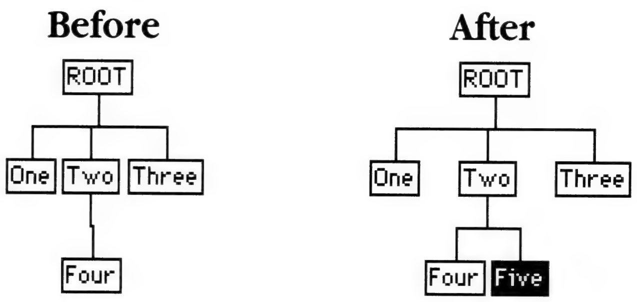


Figure 5. Before and After inserting a node into the tree

When `InsertNode` calls `ReceiveMessage`, lines 1-4 will cause `timesCalled` to be set to 1, and will prepare the tree's port for drawing. Notice that we pass it a bogus value (1313) so that the port will not get prematurely restored when we actually draw the nodes in the tree. Lines 5 and 6 then note that the parent must be resized and may need to move. The child, recall, having just been added, is already marked as needing to be resized, moved, and drawn. Line 7 calls `CalcFamilySize` on the subtree whose root is "two". CalcFamilySize will calculate all of the extents for node "five" and then modify the extents for node "two." In the process it will set their `needsResize` flags to FALSE. Note that node "four" will not have its extents recalculated, since it's `needsResize` flag is still FALSE.

At this point, line 8 runs algorithm ER on node "two", causing all three nodes to have their bounds recalculated. An addition to the ER algorithm, shown below, considers each node as it comes out of the recursion.

```
if (!EqualRect (&OldNodeRect, &newNodeRect))
  {
  if (!EqualRect (&emptyRect, &OldNodeRect))
    EraseNode (FALSE);
    this->needsDraw = eraseKidJoins = TRUE;
  }
else // if the node's children have moved, redraw it too
if (Just != kJustCenter)
  {
  if ((Orient == kBottomUp) || (Orient == kTopDown))
    {
    if (OldGChildRect.right - OldGChildRect.left !=
      newGChildRect.right - newGChildRect.left)
      this->needsDraw = eraseKidJoins = TRUE;
    }
  else
    {
    if (OldGChildRect.bottom - OldGChildRect.top !=
      newGChildRect.bottom - newGChildRect.top)
      this->needsDraw =  eraseKidJoins = TRUE;
    }
  }
if (eraseKidJoins)
  for (i = 1; i <= this->numItems; i++)
    EraseJoin (this, (CPPVisualTreeNode *)NthChild(i));
```

In this code segment, the node's new node rectangle is compared to its old one. If either the node or its children have moved, the node and its join to its parent are erased, and the node is marked as needing to be redrawn. If necessary the joins between the parent and its children are then erased.

When we pop out to lines 9 and 10 of ReceiveMessage, we set needsDraw to TRUE if the node we are considering was the one to which the message was passed; this is so that even if the addition of the child did not cause the node to move, it will redraw itself and the child which was added. Line 11 then checks to see if the node moved as a result of the child being added. If it did, it will probably cause its parent to move, so line 12 passes the kNodeAdded message to that node's parent. This process repeats itself until it reaches a node which does not move, or until it reaches the root. At each stage, the nodes which move and the lines which connect them to their children are erased by the special segment shown above.

When we finally reach the root or a nonmoving node, the recursion stops and we reach line 13. If the node needs to be drawn, line 14 draws not only that node, but the entire subtree which it is the head of. This will cause all of the nodes which have been erased and marked as needing to be drawn to draw themselves and their joins. At this point, there are no nodes in the tree which are marked as needing to be drawn. Lines 15 and 16 will then force all of the joins in the entire tree to be redrawn – admittedly this is overkill, but the cost in terms of time is practically negligible, and it makes the code much simpler.

Lines 17-25 are executed only when the tree shrinks in size, and makes sure that there are no remains of the old tree lurking about on the screen. Finally the node calls

`AdjustTree` (I will discuss the point of this routine later). It is clear that lines 14-27 will only be executed once for each operation, since once the tree is drawn, no other nodes need to be drawn. We will then step out of the recursion until we reach the top level, where lines 29-31 will restore the drawing environment to what it was before the operation started.

Now let's tackle `AdjustTree`. It is possible, during ER's execution, for some of the nodes to be given negative positions – especially in the bottom-up and right-to-left orientations. Consider a bottom-up tree which has just had a child added to its topmost node. The new node's top left corner will be given a position of `branchLength` + the node's height, which is clearly at some negative vertical coordinate, if it's parent's top left corner was at x,0. AdjustTree, shown below, simply gets the bounding rectangle of the root of the tree, checks to see if its top-left corner matches the desired top left corner of the tree, and, if it does not, forces it to move over.

```
void CPPVisualTree::AdjustTree (void)
{
  Rect tempRect;
  if (this->topNode)
    topNode->GetFamilyBounds (&tempRect);
  if ((tempRect.left != this->topLeft.h) ||
      (tempRect.top != this->topLeft.v))
    ForceMove (TRUE, this->topLeft);
}
```

One last piece of magic remains. You may have noticed that while it is the routine `DoCalcFamilyBounds` which performs algorithm ER, `ReceiveMessage` calls a glue routine named `CalcFamilyBounds`. The code for this routine is shown below.

```
void CPPVisualTreeNode::CalcFamilyBounds (Point TopLeftCorner)
{
  short  deltaH = TopLeftCorner.h - this->familyRect.left,
         deltaV = TopLeftCorner.v - this->familyRect.top;
  Point  TopRight = {this->familyRect.top + deltaV,
               this->familyRect.right + deltaH},
         BottomLeft = {this->familyRect.bottom + deltaV,
               this->familyRect.left + deltaH};
  if (!this->needsMove) return;
  this->needsMove = FALSE;
  switch (GetOrientation()) {
    case kTopDown:
    case kLeft2Right :
      DoCalcFamilyBounds (TopLeftCorner);
      break;
    case kBottomUp:
      DoCalcFamilyBounds (BottomLeft);
      break;
    case kRight2Left:
      DoCalcFamilyBounds (TopRight);
      break;
  }
}
```

When we discussed the pseudo-code for algorithm ER we mentioned that one of the parameters we passed into the

Presenting the extra-strength text editor.

If you thought previous versions of QUED/M were powerful, wait until you program with QUED/M 2.7, loaded with these new pain-relieving features:

- Integrated support for THINK Project Manager™ 6.0 & 7.0
- THINK™ debugger support
- CodeWarrior™ support (now it's easier than ever to program for the Power Macintosh®!)
- MPW ToolServer™ support
- PopUpFuncs™ support
- Frontier™ Do Script support

For quick relief, call 800-309-0355 today.

Of course, QUED/M 2.7 still has all the features that make it easier to use than any other text editor:

- Macro Language lets you automate tedious tasks
- Search and Replace through multiple unopened files and using GREP metacharacters
- File comparisons using GNU Diff
- Unlimited undos and redos
- 10 Clipboards that can be edited, saved, and printed
- Customizable menu keys
- Text folding
- Display text as ASCII codes
- Plus many more features!

Get even more relief: Try QUED/M 2.7 now for just $69! You'll get a 30-day money back guarantee too! Call now to order. 800-309-0355.

NISUS Software Inc.

QUED/M is a trademark of Paragon Concepts, Inc. All other products are trademarks or registered trademarks of their respective holders.

107 S. Cedros Ave. • Solana Beach, CA 92075 • Tel (619) 481-1477 • Fax (619) 481-6154

algorithm was the top left corner of the family. When we are drawing a tree which is oriented from bottom to top or from right to left, however, calculations are done a little differently. In a top-down or left-to-right tree we compute a node's rectangle by adding the node height to the top left corner. When we are computing the node's rectangle in a bottom-up tree, it simplifies the computation in ER to assume that we are given the *bottom* left corner and can subtract the height from that point to get the top left corner. Similarly, when we are drawing right-to-left, it is simpler to assume that we are given the top right corner and can subtract the width of the node in order to get the top left corner. The glue code in `CalcFamilyBounds` manufactures the top right and bottom left points for these two cases, then calls `DoCalcFamilyBounds` to run algorithm ER on the subtree. Once the point is passed in, its sense (ie. is it the top-left, bottom-left, top-right corner) remains the same until ER exits.

That's all the magic there is, folks. By erasing and redrawing only the nodes which move, we satisfy our minimum redraw restriction and also minimize the number of times in which a node's width and boundaries have to be recalculated.

Like CPPVisualTree, CPPVisualTreeNode has a few routines to handle direct-manipulation by the user. `DoDoubleClick` and `DoSingleClick` are virtual functions which you can customize to perform some action when the user double or single clicks on a node. `DoOnFamilyInRect` lets you perform some action if a node is partially or fully inside a specified rectangle. This is useful for setting the class' `isSelected` field to TRUE if it is inside a selection rectangle. The `PointInNode` method lets you do hit detection within a family; calling this method for the root of a tree will return a reference to any node which was clicked on in the entire tree.

CPPPackedTreeNode

There is one class remaining – the one which implements algorithm SG. Actually it implements SG+, since it makes a second pass through the tree to make sure that all of the childless nodes are properly centered between nodes with children on either side of them. Figure 6 provides a comparison of output produced by algorithm SG and SG+; note that the inverted node is properly centered by SG+, but not by SG.

LS Object Pascal

Powerful ▪ Faster ▪ Native

shipping on CD-ROM for PowerMac

now $399 was ~~$695~~

1-800-252-6479

Language Systems Corp. • 100 Carpenter Dr. • Sterling, VA 20164 • (703) 478-0181 • Fax: (703) 689-9593 • AppleLink: LANGSYS

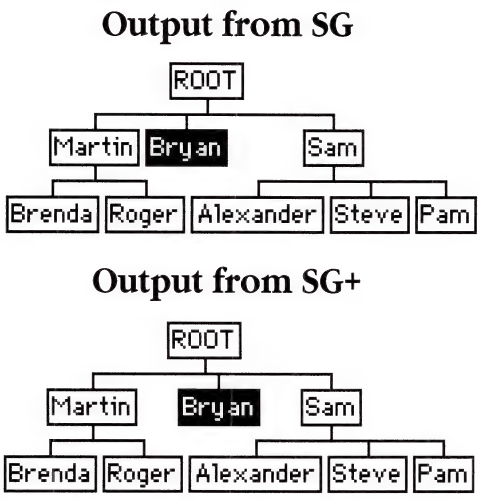


Figure 6. Comparison of SG and SG+

CPPPackedTreeNode is subclassed off of CPPVisualTreeNode. This means that it can take advantage of all the direct-manipulation, selection, etc. methods which CPPVisualTreeNode defines and only redefine the methods which receive messages and do the size and boundary calculations, as shown in its definition below.

```
class CPPPackedTreeNode : public CPPVisualTreeNode {
    friend class CPPVisualTree;
public:
      CPPPackedTreeNode (CPPObject *NodeData, CPPTree *BelongsTo,
                 Boolean becomeOwner, Boolean selected);
      ~CPPPackedTreeNode (void);
  virtual  char *ClassName (void);
  virtual  Boolean  Member (char *className);
  virtual CPPObject *Clone(void);

  virtual  void EraseNode (Boolean wholeFamily);
  virtual  void ReceiveMessage (CPPGossipMonger *toldBy,
                 short reason, void* info);
protected:
  Point  estTopLeft;

  virtual  void   DoCalcFamilySize(void);
  virtual  void   DoCalcFamilyBounds (Point TopLeftCorner);
      void   CPBRightTraverse (short level, short levelOffset);
      void   CPBLeftTraverse (short level, short levelOffset);
      void   CPBTopTraverse (short level, short levelOffset);
      void   CPBBottomTraverse (short level, short levelOffset);
      void   PostProcess (Boolean isVertical);
private:
      void   ShiftNode (Point newTopLeft);
};
```

CPPPackedTreeNode also overrides `EraseNode`. In CPPVisualTreeNode the separation between subtrees allowed us to erase a node's family by simply erasing the node's FamilyRect, but since subtrees can be 'pushed together' by SG, we have to take a more sophisticated approach – explicitly traversing the whole subtree and erasing each node individually.

CPPPackedTreeNode defines one new variable – estTopLeft – and defines six new methods – all protected or private. The four CPBxxxTraverse methods implement the SG algorithm for each different orientation, and so essentially take the place of CalcFamilyBounds in CPPVisualTreeNode. Why four instead of one? Because the calculations in SG are sufficiently involved that a single routine would very long and therefore very hard to follow. The PostProcess routine performs the "+" part of SG+ – making sure that the nodes at each level of the subtree are properly centered. PostProcess is called after DoCalcFamilyBounds positions the entire subtree. PostProcess calls ShiftNode, which simply erase a node and its join, moves it over, and redraws it. If many nodes have to be centered in this way, there is a loss of efficiency, since each one will have to be drawn twice; the price we pay for aesthetics.

Looking at the key segment of code for ReceiveMessage (shown below), shows that it is much simpler than the one in the CPPVisualTreeNode class.

```
        case kNodeAdded :
        case kNodeChangedData :
        case kChildDeleted   :
        case kChildRemoved   :
1         if (this->Root)
            {
2             ((CPPVisualTree *)this->Root)->
                                    Prepare((CPPObject *)1313L);
3           ((CPPVisualTree *)this->Root)->ForceResize (FALSE);
4           topNode->DrawNode(TRUE, FALSE);
5             ((CPPVisualTree *)this->Root)->DrawAllJoins(TRUE);
6           if ((OldFamilySize.h > topNode->familySize.h) ||
               (OldFamilySize.v > topNode->familySize.v))
              {
7             Rgn1 = NewRgn();
8             Rgn2 = NewRgn();
9             RectRgn(Rgn1, &oldFamilyRect);
10            RectRgn(Rgn2, &topNode->familyRect);
11            XorRgn (Rgn1, Rgn2, Rgn1);
12            EraseRgn (Rgn1);
13            DisposeRgn(Rgn1);
14            DisposeRgn(Rgn2);
              }
15          ((CPPVisualTree *)this->Root)->AdjustTree();
16          ((CPPVisualTree *)this->Root)->
                                       Restore((CPPObject *)1313L);
            }
          break;
```

The packed tree node class does not use recursion to limit the number of size and frame calculations as the visual tree node class does, since changes in one node can have effects on other nodes which are at the same level but in entirely different subtrees. Instead, after setting up the drawing environment in line 2, it calls ForceResize in line 3 which sets the needsMove and needsResize flags to true and calls CalcFamilySize and CalcFamilyBounds on the entire tree. As before, if any nodes move as a result, they erase themselves, their joins, and the joins of their siblings. Once everyone has recalculated their frames and extents, line 4 tells the topmost node to draw its entire family – which passes the 'draw' message to every node. The "redraw when necessary" criterion is preserved, however, because the needsDraw flag (which will be set to false in any node which did not move) prevents them from being drawn again. Lines 6-15 then perform the same functions as lines 13-27 in the visual tree node's ReceiveMessage method.

Guess what? We've finished talking about CPPPackedTreeNode; simple, eh? I won't go over the code which implements algorithm SG, since, as before, it follows fairly logically from the pseudo-code.

In Conclusion

That's all folks. Hopefully the discussion of tree-drawing algorithms has helped clarify things for those of you who have wondered how it's done. I've spent a lot of time getting these classes as bug-free and generally useful as possible, so if anyone finds any bugs, or finds a way to add functionality to them, please snail/e-mail me and let me know. For those of you who are still wondering how they can use all the options which these classes provide, let me suggest the following:

- Top-Down or Left-to-Right orientations:
  Great for displaying linguistic parse trees, hierarchical search spaces, computer directory structures, and corporate org. charts (not to mention all those juicy tree-based data structures!)
- Bottom-Up orientation:
  Family trees
- Right-to-Left orientation:
  Match-trees for sports competitions

Now, let's see what *you* can come up with!

Related Articles

Moen, Sven, *Drawing Dynamic Trees*, Technical report from the Department of Information and Computer Science, Linköping University, Sweden.

Radack, Gerald M., *Tidy Drawing of m-ary Trees*, Case Western Reserve University, November 1988.

Reingold, Edward M. and Tilford, John S., "Tidier Drawing of Trees", *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 2, March 1981.

Tilford, John S., *Tree Drawing Algorithms*, Master's Thesis, University of Illinois at Urbana-Champaign, 1981.

Wetherell, Charles and Shannon, Alfred, "Tidy Drawing of Trees", *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5, September, 1979.

Interface of the CPPVisualTree class

```
class CPPVisualTree : public CPPTree {
public:
  CPPObject  *lastClicked;
  Boolean    forceDraw;

        CPPVisualTree (WindowPtr itsWindow,
                Point  newTopLeft,
                orientStyle orientation = kTopDown,
                justStyle justification = kJustCenter,
                joinTypes join = kRightAngle,
                short branchLength = 25);
        ~CPPVisualTree (void);
```

EHelp

TOPICS PREVIOUS NOTES KEYWORD SEARCH

Add Multimedia Documentation To Your Applications And Now...

Build Stand-Alone Documentation With

EHelp 4.0

THE PREMIER DOCUMENT DISPLAY SYSTEM FOR THE MACINTOSH

- Integrated multimedia elements
- Flexible hot-spot linking
- Robust hypertext capabilities
- Sophisticated searching
- Easy integration with your applications
- Reads WinHelp™ and MSWord™ source
- Apple-event aware
- Function macros for inter-document jumps, control of user buttons, launching other applications, etc.
- World Ready for all Macintosh-supported languages
- Many other features

CALL **(800) 247-7890** OR **(919) 481-3517** FOR A **FREE EHELP** DEMO DISKETTE.

FSI FOUNDATION SOLUTIONS INC

FAX (919) 481-3551

EHelp 4.0 is a trademark of Foundation Solutions, Inc.
EHelp 3.0 is still available and supported.

```
virtual  Boolean  Member (char *className);
virtual  char *ClassName (void);

    void Prepare (CPPObject *caller);
    void Restore (CPPObject *caller);

    void ForceRedraw (Boolean doErase);
    void ForceResize (Boolean doDraw);
    void DrawAllJoins (Boolean doDraw);
    void ForceMove (Boolean doDraw, Point topLeft);
    void AdjustTree (void);

    void DrawDefaultJoin (CPPVisualTreeNode *FromNode,
                 CPPVisualTreeNode *ToNode);

    void SetOrientation (orientStyle newOrient);
    void SetJustification (justStyle newJust);
    void SetJoinType (joinTypes newJoin);
    void SetBranchLength (short newLength);
    void SetNodeMargin (short newMargin);

inline orientStyleGetOrientation (void) { return
                                    this->orientation;}
inline   justStyle  GetJustification (void) { return
                                    this->justification;}
inline   joinTypes  GetJoinType (void) { return
                                    this->whichJoin;}
inline   short    GetBranchLength (void) { return
                                    this->branchLength;}
inline short    GetNodeMargin (void) { return
                                    this->nodeMargin;}

virtual  void Draw (void);

virtual  Boolean  DoCommand (short commandID);
```

```
  virtual  Boolean  DoTreeClick (EventRecord *theEvent);
  virtual  void DoCut (void);
  virtual  void DoCopy (void);
  virtual  void DoPaste (void);
  virtual  void DoClear (void);
  virtual  void DoSelectAll (void);

      void GetTreeBounds (Rect *itsBounds);
      PicHandle  GetTreePicture (Boolean asQDPICT,
                                 Boolean placeInClipboard);

protected:
  WindowPtr  theWindow;
  Point      topLeft;

  Boolean    isPrepared;
  CPPObject  *preparedBy;
  GrafPtr    SavePort;

  long       lastClickTime;

  orientStyle orientation;
  justStyle   justification;
  joinTypes   whichJoin;
  short       branchLength;
  short       nodeMargin;

  virtual  void UserSpecificPrepare (void);
  virtual  void UserSpecificRestore (void);
  virtual  void DrawPoint2Point (Point FromPt, Point ToPt);
  virtual  void DrawRightAngle (Point FromPt, Point ToPt,
                 orientStyle orientation);

};
```

DragInstall makes your installer choice *easy*

Easy for your users

DragInstall's unique drag-and-drop interface makes installing your software a piece of cake for your users. And cuts down on your tech support calls.

Easy for you

DragInstall's Builder utility allows you to create complex installers in minutes not days. And without the need to learn a complicated scripting language.

Easy on your wallet

DragInstall's one-time fee of only $300.00 allows you to distribute an unlimited number of installers. No yearly renewals, no royalties, no hassle!

To find out just how easy DragInstall is, call us at
1-800-890-9880
to receive a free demo disk.
Or contact us at:

Ray Sauers Associates, Inc.
1187 Main Avenue, Suite 1B
Clifton, NJ 07011-2252 USA
Voice: 201-478-1970
Fax: 201-478-1513
AppleLink: D1922
Compuserve: 70731,2326
Internet: sauers@aol.com

INTERFACE OF THE CPPVISUALTREENODE CLASS.

```
class CPPVisualTreeNode : public CPPTreeNode {
    friend class CPPVisualTree;
public:
  Boolean  needsResize;
  Boolean  needsMove;
  Boolean  needsDraw;
  Boolean  notYetPlaced;

      CPPVisualTreeNode (CPPObject *NodeData, CPPTree *BelongsTo,
                 Boolean becomeOwner, Boolean selected);
      ~CPPVisualTreeNode (void);

  virtual  char *ClassName (void);
  virtual  Boolean  Member (char *className);

  virtual CPPObject *Clone(void);

  virtual  void DrawJoin (CPPVisualTreeNode *FromNode,
                 CPPVisualTreeNode *ToNode);
  virtual  void EraseJoin (CPPVisualTreeNode *FromNode,
                 CPPVisualTreeNode *ToNode);
  virtual  void DrawNodeData (void);

        void DrawNode (Boolean wholeFamily, Boolean forceDraw);
  virtual  void EraseNode (Boolean wholeFamily);
           void ResizeFamily (Boolean wholeFamily);
           void MoveFamily (Point *topLeft);

  virtual  void CalcNodeSize (void);
           void   CalcFamilySize(void);
           void   CalcFamilyBounds(Point TopLeftCorner);

  static long NumSelectedNodes (CPPVisualTreeNode *theNode);

        Boolean  CanDraw (void);

  inline Boolean  IsSelected (void) {return this->isSelected;}
        void SetSelect (short selectType,
                    Boolean selectFamily,Boolean doRedraw);

        void GetAnchorPoints (Point *Left, Point *Right,
                    Point *Top, Point *Bottom);

  virtual  void ReceiveMessage (CPPGossipMonger *toldBy,
                  short reason, void* info);
  virtual  void GetFamilyBounds (Rect *bounds);

        CPPVisualTreeNode *PointInNode (Point clickPt);

        void DoOnFamilyInRect (Rect *theRect,
      Boolean fullyInside, ApplyProc theProc, long param);

  virtual  void DoDoubleClick (short modifiers);
  virtual  void DoSingleClick (short modifiers);

protected:
  Boolean  isSelected;
  Point    nodeSize,
           childSize,
           familySize;
  Rect     nodeRect,   // the rectangle surrounding the node
           childRect,  // surrounds the node's immediate children
           gChildRect, // surrounds all of the node's descendants
           familyRect; // surrounds the node & all its descendants

  virtual  void     DoCalcFamilySize(void);
  virtual  void     DoCalcFamilyBounds (Point TopLeftCorner);

      short       GetNodeMargin (void);
      orientStyleGetOrientation (void);
      justStyle   GetJustification (void);
      joinTypes   GetJoinType (void);
      short       GetBranchLength (void);
};
```

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

By Eric Traut, Cupertino, CA, etraut@apple.com

Taking Extreme Advantage of PowerPC

Performance analysis and tuning tips to help your programs scream!

The Importance of Performance

By now, most people in the computer industry have received the message: PowerPC offers impressive performance gains over older CISC architectures. This is especially obvious to Macintosh developers who have already ported their software to the new architecture. Many have seen performance gains in the 200-400% range. But is this enough?

Although it is true that personal computers consistently double in speed every 18 months or so, computer users continue to demand faster and faster machines. In fact, the general public is probably more aware of computer performance now than ever before. Showdowns between such rivals as PowerPC and Intel's Pentium are frequently covered in the media, increasing public awareness further. Developers continue to create more sophisticated and CPU-intensive applications. This all boils down to the fact that software performance tuning is more important than ever.

Customers demand low-cost, high-performance machines, and Apple has staked its future on PowerPC because it believes that RISC can deliver on these promises. However, the maximum benefit of RISC can be realized only with the support and cooperation of third-party software vendors. Many developers have already taken the first step and ported their products to run native on PowerPC-based Macs. The next step involves performance analysis, tuning and, most importantly, innovation. By taking advantage of new-found speed in imaginative ways, developers can deliver dazzling software that is easier and more enjoyable for users.

“Many have seen performance gains in the 200%-400% range. But is this enough?”

What Is Performance?

Although this article concentrates on CPU execution time, it should be noted that performance can (and should) be considered at many different levels. For example, *perceived performance* often has little or no relation to actual performance. Perceived performance of an operation can be increased simply by providing user feedback. This may include spinning beach-ball cursors, status bars, or other visual progress indicators. Along with perceived performance comes the concept of *responsiveness.* How quickly does a program react to a user event in some perceivable way? Both perceived performance and responsiveness are important considerations when designing and implementing Macintosh software.

Unlike perceived performance, *actual* performance can be

Eric Traut – Eric works for Apple Computer, Inc., and has been involved with PowerPC software for the past three years. He was one of the main engineers responsible for performance analysis and tuning of the original Power Macintosh system software release and has worked for the past year on second-generation 680x0 emulation technology. Eric has assisted hundreds of Macintosh developers worldwide in their efforts to port to the new RISC platform and tune their code to take advantage of PowerPC.

Online & CD-ROM Publishers:

BroadCast™

Electronic Distribution That Works!

Expand your market beyond the confines of traditional distribution methods and channels. Now you can offer your customers **software on demand**, with **BroadCast™**.

Sell Your Software Online

Convert your online presence from a support/marketing cost center to a profitable distribution channel. Perhaps you already post "crippled" demos online, but have no means of follow-through, since the demos are downloaded anonymously. You can *hope* that subscribers are favorably impressed; *hope* that they elect to purchase live software at some future date; and *hope* they complete and mail registration cards.

BroadCast taps the enormous potential of online services as a sales medium, enabling you to convert passing interest into direct sales, by offering your products to subscribers at the precise moment when their motivation is at its peak.

Sell Your Software on CD-ROM

CD-ROM is an attractive storage medium. **BroadCast** transforms it into your **most profitable sales medium**. With **BroadCast**, you can publish your entire software line, with demos of each product, on a single CD. When a customer purchases one of your products, he or she is also exposed to your other products, and can unlock them, on demand, with a single telephone call.

BroadCast is easy to implement In seconds, **BroadCast** will securely compress and encrypt your product, and embed it in an unlocker application that can be publicly distributed. To buy your product, a customer simply runs the unlocker. A unique control number is displayed, together with instructions and the telephone number of your sales desk. Your sales representative processes a credit card transaction and captures registration information. Using software we provide, the sales representative then generates a unique unlocking password for the customer. When the customer enters that password, the product is unlocked.

BroadCast is secure Our patent-pending technique takes a "thumbprint" of the user's system, and always requires a unique password.

BroadCast is inexpensive Why give up 50 points or more (plus co-op) to a reseller, when you can sell direct at a fraction of the cost! No receivables, no cost of goods, no freight. Since customers try before they buy, you'll likely have no returns, either!

SNA

SNA, Inc.
2200 NW Corporate Blvd.
Boca Raton, FL 33431
Tel (407) 241-0308 • FAX (407) 241-3195

objectively measured. For many types of software, actual performance can be measured with a stop watch. Other types of software may be optimized for greater bandwidth, lower latency, or a greater number of events per unit time. For example, QuickTime performance is often measured in frames per second, whereas bandwidth and latency are more important for networking software. Before tuning can begin, it is important to understand which performance measurement is most appropriate for the piece of software being analyzed. Without this understanding, there would be no way to determine if performance tuning efforts were successful.

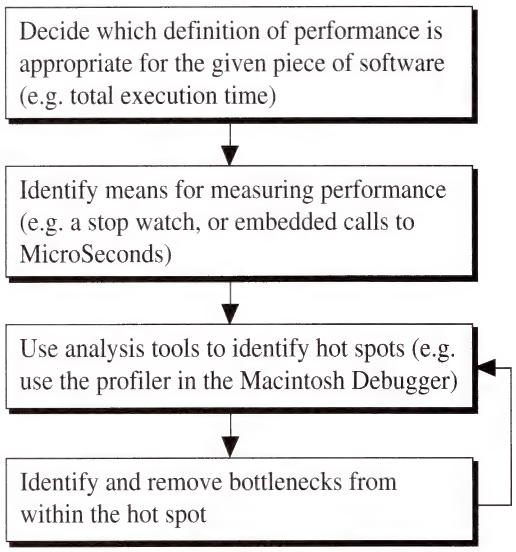

Decide which definition of performance is appropriate for the given piece of software (e.g. total execution time)
Identify means for measuring performance (e.g. a stop watch, or embedded calls to MicroSeconds)
Use analysis tools to identify hot spots (e.g. use the profiler in the Macintosh Debugger)
Identify and remove bottlenecks from within the hot spot

Figure 1: Performance analysis and tuning process

Performance Tuning

Once the objectives for analyzing a piece of software are established, and a means for measuring performance has been identified, tuning can begin. This usually requires an iterative process involving a few simple steps. Tuning efforts should proceed until performance objectives are met or further tuning would result in little or no improvement.

The first step of the process typically requires the use of performance tools to identify *hot spots* within the code. These hot spots represent portions of the program which are critical for performance. Developers often assume they know where the hot spots are in their programs. Though they are often correct, a bad assumption can result in lost time and energy. In most cases, profiling and tracing tools will quickly and accurately identify hot spots.

How should you decide whether a piece of code is worth tuning? Ask yourself how much overall performance will increase if the tuning results were best case. For example, if a particular subroutine accounts for 4% of the total execution time, the best that can be done by tuning this subroutine is an overall speedup of 4%. If this were the case, the subroutine would probably not be worth tuning unless all other efforts had failed to achieve the performance goal.

Once a hot spot is identified, efforts should turn to detecting and removing *bottlenecks*, or ways in which the hardware is being used inefficiently. Bottlenecks come in many different forms, and one of the biggest challenges to performance tuning is sniffing them out and removing them. Much of the remainder of this article discusses various types of bottlenecks and ways to remove them.

PERFORMANCE ANALYSIS TOOLS

Keep in mind that removing a bottleneck which is not within a hot spot will have little or no effect on overall performance. Identifying hot spots can be challenging, but tools exist to help make the job easier. There are two main classes of analysis tools: *tracers* and *profilers*.

Tracers are used to record a series of events, often time-stamping each event as it is logged. Tracers are most useful when event ordering is of importance or when a distribution of timings is needed. The disadvantage to tracers is resource consumption. Because events are continually logged, traces can take up large amounts of RAM and disk space. This volume of data is not only difficult to store, but also difficult to analyze. However, tracers still have their use. For example, a tracer was used by Apple engineers to identifying common system call chains. This information was then used to determine which portions of the system to port to PowerPC first.

Profilers also record information throughout the execution of the program, but no ordering information is maintained. Rather, information is totaled or averaged throughout the profiling session. This means that profiler output does not consume as much space as that of tracers, and the resulting data is often easier to understand.

A number of profiling tools are available to Mac developers. Most compilers provide options to generate profiling code which collects function-level statistics at runtime. These statistics usually include an execution count and an average execution time for each profiled routine.

Another very useful profiling tool is built into Apple's Macintosh Debugger for PowerPC. This debugger, which can be used in a one or two-machine environment, makes use of a mechanism called *adaptive sampling profiling*, or ASP. When enabled, ASP periodically samples the program counter. These sampled addresses are mapped to *buckets* which represent ranges of addresses. Each time the program counter falls within a bucket, the bucket's counter is incremented. When a counter reaches a certain level, the ASP splits the bucket into two smaller buckets. This allows the tool to adapt to the code being profiled, providing higher resolution for hot spot areas. At the end of an ASP session, the tool attempts to map all address ranges to code symbols. It also produces a report which details how much time is spent within the system software as well as the percentage of time spent in emulated code. All of this information is useful in locating hot spots and removing bottlenecks.

NEW! Version 2.0 - Supports PPC & Fat Binary

PatchWorks™

Builds Updaters Without Programming

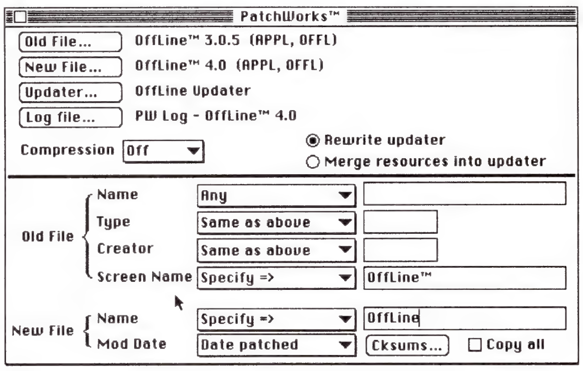
PatchWorks has many options, but only one function: to create updater applications for distribution via public channels (e.g., online services).

Before the advent of ***PatchWorks***, creating an updater was a project in itself, one that consumed valuable programmer time which could more profitably be spent on revenue-producing projects.

With ***PatchWorks***, you create an updater in minutes. Since there's no coding or scripting, no bugs are introduced. Just fill in a dialog, and ***PatchWorks*** does the rest!

Distribute updaters frequently to reflect maintenance releases, and watch your tech support and fulfillment costs fall dramatically.

Most important, your customers will know you care.


PatchWorks™

Old File...	OffLine™ 3.0.5 (APPL, OFFL)
New File...	OffLine™ 4.0 (APPL, OFFL)
Updater...	OffLine Updater
Log file...	PW Log - OffLine™ 4.0

Compression: Off

[✓ Rewrite updater] / Merge resources into updater

Old File	Name	Any	
	Type	Same as above	
	Creator	Same as above	
	Screen Name	Specify =>	OffLine™
New File	Name	Specify =>	OffLine
	Mod Date	Date patched	Cksums... ☐ Copy all

Features

- Works with apps, INITs, cdevs, fonts, drivers, etc.
- Now supports PPC, fat binary, & 4D apps
- Customizable user interface
- Updaters support multiple old versions
- Resource compression (diffing) produces small updaters
- Preserves personalization data (name, serial #, etc.)
- Updater distribution is **unrestricted & royalty-free**

Pricing: Begins at $195. Call for more information.

SNA

SNA, Inc.
2200 NW Corporate Blvd.
Boca Raton, FL 33431
Tel (407) 241-0308 • FAX (407) 241-3195

Removing Hot Spots

Once a hot spot is found, an attempt should be made to remove the hot spot completely. This can be done by analyzing program flow. How is the hot spot being entered? Is the routine which contains the hot spot being called more often than necessary? Is there an algorithmic change which will alleviate some of these calls (e.g., a caching mechanism)?

Some of these types of problems are easy to remedy. Take, for example, the following code snippet:

```
// this code draws a column of strings within the window
while (stringIndex < stringCount) {
  SetPort(theWindow);
  TextFont(geneva);
  TextSize(12);
  ForeColor(blackColor);
  MoveTo(h, v);
  v += 14;
  DrawText(stringArray[stringIndex++]);
}
```

This loop has a number of performance problems. First, it is clearly unnecessary to set the port, change the text characteristics, and set the drawing color each time through this loop. Second, if the number of strings is sufficiently large, it is unlikely that all of the strings will be visible within the window at one time. DrawText is an expensive routine, and it should be avoided if it can be determined that the text will fall outside of the window's clip region.

A profile of the above code would probably flag various QuickDraw routines as hot spots. When Mac toolbox routines show up high on the list of hot spots, the only recourse is to attempt to call these functions less frequently.

Toolbox hot spots sometimes occur when code makes implicit assumptions about processor speed – assumptions which may no longer be true on PowerPC processors. For example, take a look at the following loop:

```
// calculate vectors until complete or user presses command-period
while (!done && !canceled) {
  done = CalculateVectors(vectorArray, kVectorsPerCall);
  canceled = DetectCancelEvent();
}
```

This code contains a constant *kVectorsPerCall* which is used to control the number of vectors processed before checking for a user event. Although this constant may have been appropriate on a Mac Plus, it is very likely too small for a Power Mac. Checking for user events is only necessary every three to six ticks. This code may be spending a large percentage of its time in the Event Manager checking for a command-period. The code should probably be changed to something like the following:

```
// calculate number of vectors until complete or user presses command-period
prevTicks = TickCount();
while (!done && !canceled) {
  done = CalculateVectors(vectorArray, kVectorsPerCall);
  if (TickCount() - prevTicks >= kUserInteractionTicks) {
    canceled = DetectCancelEvent();
    prevTicks = TickCount();
  }
}
```

This code will dynamically adjust to the speed of the processor. A similar technique should be used for the main event loop of an application. Many native PowerPC programs spend more time than necessary within WaitNextEvent because of implicit assumptions about processor speed. There is no need for the Event Manager to become a hot spot within your program.

Within some types of code, it is difficult to determine why a particular piece of code is a hot spot. This is especially true for C++ where a simple variable declaration can result in the execution of a relatively complex and costly constructor routine. Operations which are normally quite simple (e.g., addition) can be overridden with a costly replacement. These features of C++ offer incredible flexibility, but they can easily create hot spots which are difficult to track down. Use these features carefully.

“It appears we have made the code longer, but the resulting machine code will be shorter and very likely faster.”

Identifying and Removing Bottlenecks

Unlike Mac toolbox routines, most functions within your program are entirely under your control. If there is a good reason for a particular function to contain a hot spot, then the next step is to remove as many bottlenecks as possible.

Below are some of the most common bottlenecks with discussions of how to remove them. This is by no means an exhaustive list of all types of bottlenecks. Hopefully, I have touched upon most of the common performance hits.

Note that all examples are given in C. On RISC platforms, well-written C code compiled with a good optimizing compiler is usually as optimal (sometimes more optimal) than hand-written assembly code. Does this mean it is not important to learn the basics of the PowerPC instruction set? The answer is unfortunately “no” if you wish to become good at performance tuning. One of the most effective ways to track down bottlenecks is to use a disassembler. The disassembled code listings will provide clues as to how the compiler interpreted the original high-level code and often suggest ways in which the original code can be improved.

Picking a Better Algorithm

The first thing to consider when examining a hot spot is whether the chosen algorithm is appropriate. Choice of algorithms should take into consideration both the expected data size and the precision of the operation. Computer science courses teach about the importance of selecting algorithms which typically require less work, using the language of “order *n*” [where the notation is $O(n)$] to describe the expected amount of work in terms of the number of elements being

Simply the best GUI Building/Event Managing libraries

Tools Plus™ 2.5

Tools Plus gives you the routines you need to create a professional looking user interface. Then we make it work. It's that simple.

- For THINK C and Symantec C/C++ (5.0.4 and later) or THINK Pascal
- Over 170 high-powered "set and forget" routines that automate and enhance: event handling, windows, the tool bar, floating palettes, cursors, buttons, picture buttons, scroll bars, menus (pull-down, hierarchical and pop-up), list boxes, fields, Edit menu, clipboard, Dynamic Alerts, and more...
- Easy to learn and easy to use
- Substantial code reduction
- Dramatic code simplification
- Significantly less debugging
- System 6 and 7 compatible
- For novice, intermediate and advanced programmers
- Runs fast; needs little memory or disk space
- Safer than toolbox routines
- No royalties

OK

Water's Edge Software

Box 70022 • 2441 Lakeshore Road West
Oakville, Ontario • Canada L6L 6M9
Phone: (416) 219-5628
CompuServe: 73424,2507
Internet: 73424.2507@CompuServe.com

Tools Plus for C/C++ or Pascal only $149 US or $199 US for both.
(We accept VISA and American Express. Add $10 for shipping.)

Language
○ C/C++
○ Pascal
● Both

Size: 9 10 ● 12 14 18 24 36

☒ Saves Time
☒ Saves Money
☒ Easy to Use

Free Evaluation Kit:
CompuServe GO MACDEV, C & Pascal library, file name: TP252.SEA
AppleLink Software Sampler/Software Collection/Programmer Tools/Tools Plus
Disk also available by mail.

worked on. These courses teach that it's better to choose O(log *n*) algorithms over O(*n*) and O(*n*) over O(n^2). Keep in mind, however, that "more expensive" algorithms may be less expensive if *n* is constrained to a certain limit.

Also remember that operations which result in more precision usually, but not always, take more time. For example, calling a library routine to compute a cosine may return 24 bits of precision when your program only really needs 8 bits. In this case, it may be best to write a cosine approximation function and avoid the library call.

Time critical routines should avoid calling extremely general functions. Take, for example, CopyBits. This routine has been optimized for certain common cases, but still needs to take into account alignment, transforms, colorizing, clipping, pixel resizing, and various other GrafPort state. Code which copies between two off-screen buffers can often make certain assumptions about those buffers (e.g., they are both eight-byte aligned, have the same background colors, and are the same pixel depth). A custom routine for copying between these buffers would probably be more efficient than calling the generic CopyBits routine.

Care should be taken when replacing library or toolbox calls – only do this when performance demands it and the benefits are obvious. For example, QuickDraw routines are handled in hardware on some graphics accelerator boards. Avoiding a QuickDraw routine may actually be slower in this case.

Register Usage

Most computers today make use of a simple memory hierarchy to compensate for the inherent speed limitations of larger storage media. On today's Power Macs, the memory hierarchy consists of secondary storage (i.e., virtual memory backing store), main memory (DRAM), second-level cache (SRAM), on-chip first-level cache(s), and processor registers. This hierarchy has been optimized to take advantage of "typical" memory access patterns. It is important to tune software so that it can take maximum advantage of the memory hierarchy and avoid costly access delays.

The PowerPC architecture, like many RISC architectures, defines 32 general purpose (integer) registers and 32 floating point registers. Performance is very dependent on the efficient use of these registers. Compilers play the most important role in optimizing the use of registers, but there are a number of techniques which can be employed by programmers to improve register usage.

Large, complex routines often contain many local variables. When compilers cannot fit all variables within processor registers, they are forced to *spill* them to memory. Routines should be kept simple enough to avoid this occurrence wherever possible.

Most RISC compilers perform *live variable analysis* to determine which variables contain live information at which point within the function and *register coloring* to efficiently assign these variables to machine registers. These techniques often allow a single register to be used for multiple local variables. Live register analysis and register assignment are complex operations, and some compilers are better than others. Programmers can help improve register usage by defining local variables only within the scope in which they are needed. Take, for example, the following piece of code:

```
void UnoptimizedRegUsage(long *a, long *b)
  long   temp;
  if (*a > *b) {
    // swap values if necessary
    temp = *a; *a = *b; *b = temp;
  }
  *a = MungeNumbers1(*a, *b);
  *b = MungeNumbers2(*a, *b);
```

All TEXT is not created equal.

Is your application taking *cut & paste* a little too literally when it creates text? Are you tired of living in a monotype world? **PAIGE**™, our text & page layout programming library, is the ultimate cross-platform solution.

PAIGE provides the most sophisticated features/functions in the business. These include:

- ***Stylized Text***
- ***Shapes & Containers***
- ***Text Wrapping***
- ***Embedded Objects***
- ***Hypertext Links***
- ***Virtual Memory***
- ***Style Sheet Support***
- ***Multi-Level "Undo"***
- ***Royalty Free***

So why should you join the **PAIGE** revolution? TIME. Most programmers don't have time to reinvent the wheel.

PAIGE was designed using no global variables and machine specific code has been isolated into two small source files. This strategy allows you to move your application to other operating systems or platforms by changing only platform specific code while maintaining full data and application compatibility.

Join the hundreds of major software publishers using **PAIGE** as their total text solution. For complete technical/pricing summary contact **DataPak Software** at 800-327-6703 or 206-573-9155.

Macintosh • Power Macintosh • Windows

```
  if (*a > *b) {
    // swap values if necessary
    temp = *a; *a = *b; *b = temp;
  }
  *a = MungeNumbers1(*a, *b);
  *b = MungeNumbers2(*a, *b);
}
```

This code makes use of a temporary variable to swap the values pointed to by *a* and *b*. Notice that although the variable *temp* is used twice, it is not live between the two uses, nor is it needed after its final use. The function also has a second performance problem. All references to the input values are made by dereferencing *a* and *b*. If we know it is not important to update **a* and **b* each time, we can store these values in temporary variables. It is often difficult or impossible for compilers to make this assumption. The new code would then look like this:

```
void OptimizedRegUsage(long *a, long *b)
  long   localA = *a, localB = *b;
  if (localA > localB) {
    // swap values if necessary
    long temp;
    temp = localA; localA = localB; localB = temp;
  }
  localA = MungeNumbers1(localA, localB);
  localB = MungeNumbers2(localA, localB);
  if (localA > localB) {
    // swap values if necessary
    long temp;
    temp = localA; localA = localB; localB = temp;
  }
  *a = localA = MungeNumbers1(localA, localB);
  *b = MungeNumbers2(localA, localB);
}
```

It appears we have made the code longer, but the resulting machine code will be shorter and very likely faster. Most important, the resulting code contains fewer loads and stores. Even if cache misses are unlikely (as in this case where we are accessing the same addresses over and over again), high-end processors like the 604 will be able to execute multiple arithmetic operations in the same cycle, whereas loads and stores are limited to one per cycle.

Due to the PowerPC runtime architecture, global and local static variables require special attention when used within a performance-critical piece of code. These classes of variables are stored within a program's data section and are referenced through the TOC (table of contents). Reading a global variable involves accessing its address by looking it up in the TOC, then dereferencing this address to get to the data. If a global or static local is used multiple times throughout a function, most compilers will generate code which performs the TOC lookup only once. However, because the compiler does not know if the global will be required or changed by other portions of the program (which could be running asynchronously to the current code), it must always perform the second dereference. If you know that the global is not needed or changed by other functions throughout the execution of the current procedure, it is often more optimal to assign the global to a local variable. Take, for example, the following code snippet in which a single global variable is used three times:

```
for (i=0; i<kLoopCount; i++) {
  SetVectorOrigin(gCurrentH, v);
  CalcVector(gCurrentH, v + gCurrentH);
  gCurrentH += 10;
}
```

We can tighten up the inner loop by changing the code to this:

```
short    localH = gCurrentH;
for (i=0; i<kLoopCount; i++) {
  SetVectorOrigin(localH , v);
  CalcVector(localH, v + gCurrentH);
  localH += 10;
}
gCurrentH = localH;
```

Cache and Virtual Memory Usage

Poor register usage can slow down a program by a factor of two to four. Poor cache usage is much worse and can slow down a program by five to ten times. Misuse of virtual memory can cause code to run thousands of times slower yet. It is therefore very important to be aware of cache and VM usage in time critical pieces of code.

Both caching and virtual memory rely on *spatial* and *temporal locality*. Spatial locality means that if a memory address has just been accessed, it is likely that other memory addresses nearby will be accessed. Temporal locality means that if a memory address is accessed, it is likely that the same address will be accessed within a small period of time. The key to tuning for caches and VM is to increase both types of locality within your code. Keep in mind that a single virtual memory page on today's Power Macs is 4K, and most PowerPC processors handle caching on a 32-byte basis.

There are three types of memory accesses which occur during the execution of most code: data loads, data stores, and instruction fetches. All of these accesses require the processor to perform *address translation* to turn the effective address into a physical address. To speed up this operation, processors make use of a *translation look-aside buffer* (or TLB for short). TLB entries are allocated on a per-page basis. This means code which erratically accesses many different pages (i.e., code with poor locality) will pay the penalty for "missing" in the TLB. Each TLB miss results in a reload which requires on the order of 10 to 50 cycles depending on the processor. This type of code is also likely to cause more page faults, each of which can take hundreds of thousands of cycles to resolve.

So, if locality is important, how can it be improved? Here is a piece of code which has poor data locality:

```
char myArray[kMaxY][kMaxX];
// initialize array
for (x=0; x<kMaxX; x++)
  for (y=0; y<kMaxY; y++)
    myArray[y][x] = x;
```

In this example, the doubly-nested loop traverses a two-dimensional array. Note that the inner loop increments *y*, the left-most array index. Unfortunately for this piece of code, C lays out arrays such that elements indexed by the right-most index are contiguous in memory. Each time we go through the inner-most loop, we are moving *kMaxX* bytes ahead in memory. By simply switching the two *for* statements, the code will access contiguous bytes each time through the loop.

"This means...that the optimized blitting loop...would be 30 to 50 times slower if care were not taken to align the accesses."

This example was fairly trivial, but it provides a taste for locality optimizations. In general, it is best to access consecutive addresses of memory in ascending order.

Within the PowerPC architecture, only a few instructions (loads, stores and a few cache-related operations) can result in a data memory access. However, the processor must perform an instruction fetch for *every* instruction. Like data, instructions are cached to speed up fetches, so code locality is also important for performance.

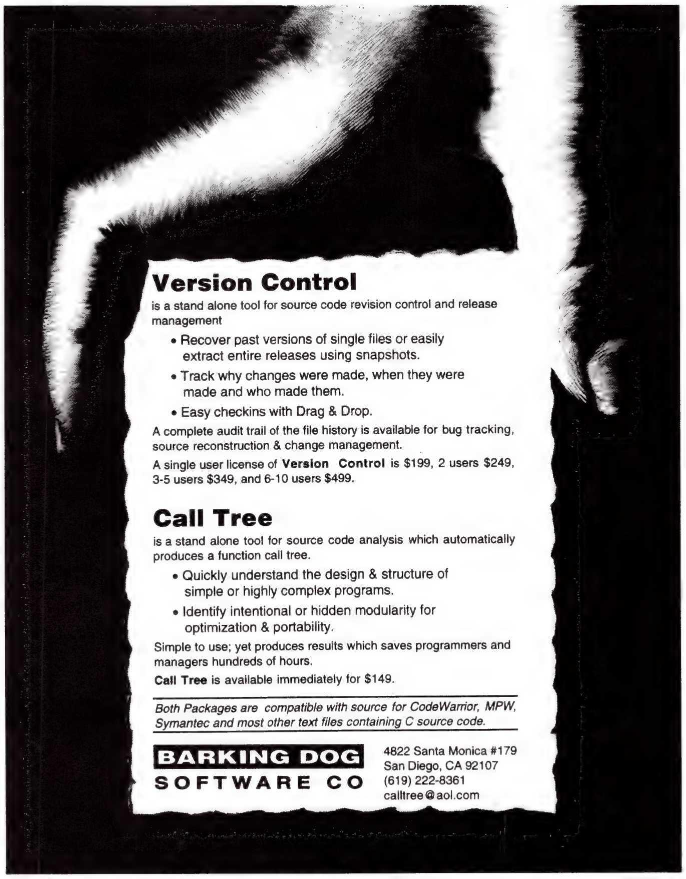
One of the most obvious ways to increase code locality is to make sure that functions which call each other are logically near each other. This increases the likelihood that they lie on the same page. This can be done at the source code level, since most compilers and linkers preserve the order in which functions are implemented within the source.

More subtle locality issues come in to play within functions. Here, page locality is not so much of an issue, but cache locality can be. Take a look at the following piece of code:

```
myHandle = NewHandle(sizeof(DataStructure));
if (myHandle == NULL) {
  // complex tear-down operation
} else {
  // initialize data structure
}
```

This piece of code is testing for an exceptional case (i.e., not enough memory to allocate the handle) and handling it appropriately. However, exceptional cases, by definition, do not occur most of the time. This means it would be better to place the exception handling code someplace else – where it is not taking up local cache space. In the above example, this can be easily accomplished by negating the conditional and swapping the two branches of the *if-else* statement.

Most real functions are more complex and contain a number of exception-handling sections. For dealing with these efficiently, I suggest using the exception macros presented by Sean Parent in the Aug. 1992 issue of **develop**. These macros are defined in



Version Control
is a stand alone tool for source code revision control and release management
- Recover past versions of single files or easily extract entire releases using snapshots.
- Track why changes were made, when they were made and who made them.
- Easy checkins with Drag & Drop.

A complete audit trail of the file history is available for bug tracking, source reconstruction & change management.
A single user license of **Version Control** is $199, 2 users $249, 3-5 users $349, and 6-10 users $499.

Call Tree
is a stand alone tool for source code analysis which automatically produces a function call tree.
- Quickly understand the design & structure of simple or highly complex programs.
- Identify intentional or hidden modularity for optimization & portability.

Simple to use; yet produces results which saves programmers and managers hundreds of hours.
Call Tree is available immediately for $149.

Both Packages are compatible with source for CodeWarrior, MPW, Symantec and most other text files containing C source code.

BARKING DOG SOFTWARE CO
4822 Santa Monica #179
San Diego, CA 92107
(619) 222-8361
calltree@aol.com

Attention Scripters!

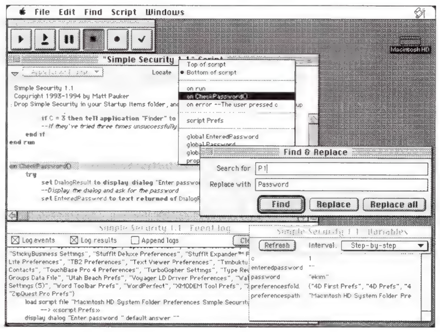
ScriptWizard

...the essential scripting tool

ScriptWizard™ brings professional script editing, testing and debugging facilities to AppleScript™. Improve your scripting productivity with this powerful and intuitive tool!

4-1/2 mice - MacUser UK
4 stars - MacWorld UK

- **Watch Variables**
  Variable-watcher shows a complete list of variables and script properties.
- **Find & Replace**
  An irreplaceable tool for complex script development and modification.
- **Step-Execution**
  Step-execution of scripts allows you to step through scripts one Apple event at a time.
- **Easy Script Navigation**
  The "locate" pop-up shows all properties and handlers defined in your script, and instantly jumps to the one you choose.



```
 Simple Security 1.1
 Copyright 1993-1994 by Matt Pauker
 Drop Simple Security in your Startup Items folder, and

        if C = 3 then tell application "Finder" to
        --If they've tried three times unsuccessfully
    end if
end run

on CheckPassword()
    try
        set DialogResult to display dialog "Enter passwo
        --Display the dialog and ask for the password
        set EnteredPassword to text returned of DialogR
```

Search for: P1
Replace with: Password

```
"StickyBusiness Settings", "StuffIt Deluxe Preferences", "StuffIt Expander™ P
Lite Preferences", "TB2 Preferences", "Text Viewer Preferences", "Timbuktu
Contacts", "TouchBase Pro 4 Preferences", "TurboGopher Settings", "Type Re
Groups Data File", "Utah Beach Prefs", "Voyager LD Driver Preferences", "Wa
Settings (5)", "Word Toolbar Prefs", "WordPerfect", "XMODEM Tool Prefs", "
"ZipQuest Pro Prefs"}
    load script file "Macintosh HD:System Folder:Preferences:Simple Securit
        --> «script Prefs»
    display dialog "Enter password:" default answer ""
```

```
c                   1
enteredpassword     ""
password            "ekim"
preferencesfold     {"4D First Prefs", "4D Prefs", "4
preferencespath     "Macintosh HD:System Folder:Pre
```

"Its debugging capabilities are a 'must have' for any serious scripter"
- Dan Shafer: Author, Consultant and Scripting Advocate

only
$99

FULL MOON SOFTWARE

US Offices:
P.O. Box 700237 San Jose, CA 95170-0237
Phone: (408)253-7199 Fax: (408)252-2378
European Offices:
P.O. Box 116 ST ALBANS, Herts, AL12RL, UK
Phone: +44 727 844232 Fax: +44 727 856139

Exceptions.h which is provided by Apple on the E.T.O. disk. They make use of the C *goto* statement so all exception handling code can be placed at the end of the function.

In any case, attempts should be made to make the "normal" code paths as contiguous and linear as possible.

Video Buffer Accesses

Although not recommended by Apple for most applications, direct access to the video buffer gives certain programs the added video performance they need for smooth animation. Accesses to Power Mac video buffers are different from normal memory accesses in one very important way. Most of the Power Mac RAM is mapped copy-back-cacheable. This means that a store operation writes its data to the first-level cache, but not immediately to memory. Main memory is updated only when the dirty cache block is replaced within the cache.

If the video buffer were mapped copy-back-cacheable, pixels would appear sporadically on the screen. For this reason, video buffers are marked either write-through-cacheable or non-cacheable. Either way, stores to video are immediately written to the buffer. Many such stores in a row will fill up the on-chip store queues and stall the processor until pending data transfers are complete. A single store to video can take from 6 to 18 cycles, depending on the processor clock rate and the speed of the video subsystem. Ideally, programs could take advantage of these stalls to execute arithmetic operations which do not involve loads or stores. Unfortunately, interleaving blitting and calculations is extremely difficult.

If code is stuck with these multi-cycle stalls, the best we can do is to make optimal use of each store. With integer registers, it is possible to store four bytes at a time. By storing data from the floating point registers, the code can store eight bytes at a time. Take, for example, the following blitting loop which performs pixel smearing, effectively doubling the width of the image as it draws it to the video device:

```
// Copy scan line from off-screen buffer to screen, doubling the width
// of the image in the process. Assumes 8-bit video.
void DoubleCopyPixels1(unsigned short *src, unsigned long
*dest)
{
  long column;
  for (column=0; column<kRowBytes/2; column++) {
    unsigned long pixels;

    pixels = *src++;
    *dest++ = TWO_TO_FOUR(pixels);
  }
}
```

If we can assume the destination is eight-byte aligned, and kRowBytes is divisible by four, then we can rewrite the code:

```
// Copy scan line from off-screen buffer to screen, doubling the width
// of the image in the process. Assumes 8-bit video.
void DoubleCopyPixels2(unsigned short *src, double *dest)
{
  long column;
  for (column=0; column<kRowBytes/4; column++) {
    double          eightPixels;
    unsigned long   pixels;

    pixels = *src++;
    ((unsigned long*)&eightPixels)[0] = TWO_TO_FOUR(pixels);
    pixels = *src++;
    ((unsigned long*)&eightPixels)[1] = TWO_TO_FOUR(pixels);
    *dest++ = eightPixels;
  }
}
```

Once again, it appears we have made the code longer and more complex. In fact, we are doing three stores each time through the loop now instead of one. Luckily, two of these stores are to a cacheable area of RAM, so they are not very expensive. We more than make up for the added complexity by doubling our bandwidth to the screen. Both QuickDraw and QuickTime make use of this technique to accelerate common blitting operations.

Data Alignment

All PowerPC instructions fall on four-byte boundaries, but data is byte-addressable. Most processors are optimized for accesses which are aligned to the data length, i.e., doubles should be aligned to eight bytes, longs to four, etc. PowerPC *C* compilers typically default to this type of alignment for structures. For backward compatibility, however, it is often necessary to use 68K-style alignment which requires a maximum of two-byte alignment within structures.

The standard universal headers released by Apple include compiler directives which force 68K alignment for old toolbox data structures. All new system data structures defined by Apple will use PowerPC alignment. Developers should also attempt to migrate toward natural PowerPC alignment wherever possible. For compatibility reasons, this may be impossible for structures which are stored to disk or sent over the network. However, most internal data structures can be changed from 68K to PowerPC alignment with no undesirable side effects (except for, perhaps, a slight increase in memory usage).

What is the cost of using misaligned data accesses? This depends on the instruction stream and the processor. Most PowerPC processors have a load latency of one cycle, i.e., the data being loaded cannot be used for one cycle after the load executes. This latency typically doubles or triples for misaligned accesses. Accesses are even more expensive when they cross cache-block boundaries. Note that this will never occur for properly aligned data. On the 601, accesses which cross page boundaries are even more expensive; they force an alignment exception which is handled by a low-level software exception handler.

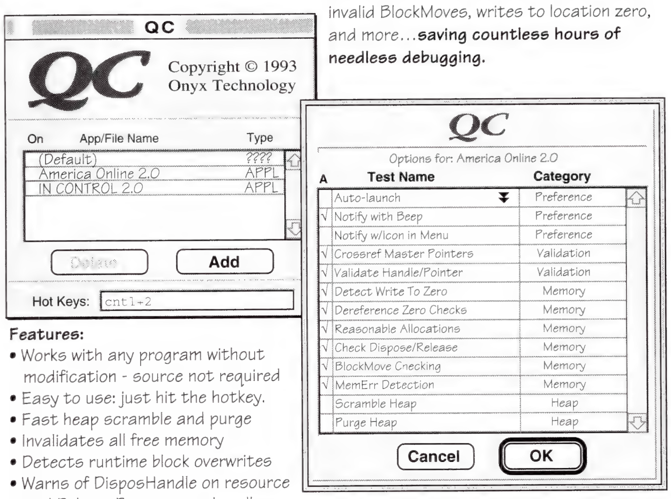
Extra special care should be used when accessing floating point values (either doubles or floats). Some processors, including the 603 and 604, force an alignment exception when floating point loads and stores access a misaligned value. This means, for example, that the optimized blitting loop shown in the previous section would be 30 to 50 times slower if care were not taken to align the accesses.

QC

NOW SHIPPING

QC: the **Macintosh Testing solution.** Subject your code to **brutal** stress conditions to **make it break** consistantly.

*Or use QC during the development cycle to casually detect block boundary overwrites, invalid BlockMoves, writes to location zero, and more...***saving countless hours of needless debugging.**


QC

QC Copyright © 1993 Onyx Technology

On	App/File Name	Type
	(Default)	????
	America Online 2.0	APPL
	IN CONTROL 2.0	APPL

Delete | Add

Hot Keys: cntl+2

QC

Options for: America Online 2.0

A	Test Name	Category
	Auto-launch	Preference
√	Notify with Beep	Preference
	Notify w/Icon in Menu	Preference
√	Crossref Master Pointers	Validation
√	Validate Handle/Pointer	Validation
√	Detect Write To Zero	Memory
√	Dereference Zero Checks	Memory
√	Reasonable Allocations	Memory
√	Check Dispose/Release	Memory
√	BlockMove Cnecking	Memory
√	MemErr Detection	Memory
	Scramble Heap	Heap
	Purge Heap	Heap

Cancel | OK

Features:
- Works with any program without modification - source not required
- Easy to use: just hit the hotkey.
- Fast heap scramble and purge
- Invalidates all free memory
- Detects runtime block overwrites
- Warns of DisposHandle on resource and ReleaseResource on handles.
- Validates BlockMove destinations
- Sophisticated heap verification
- Powerful API for precision control

30 DAY
NO QUESTIONS
ASKED
MONEY BACK
GUARANTEE

Onyx TECHNOLOGY

ONYX TECHNOLOGY
7811 27th Ave W.
Bradenton, FL 34209

$99 SPECIAL INTRODUCTORY PRICE

Tel: 813.795.7801 **Fax:** 813.792.5152 **AOL:** OnyxTech **ALink:** D2238 **CIS:** 70550.1377

Data Types

The choice of simple integer data types can affect performance. On early 68K processors, 16-bit values and operations were often faster than 32-bit. Just the opposite is true on PowerPC. The use of signed short and signed char operations can force the compiler to emit extra, often unnecessary, sign-extension instructions. Take, for example, the following piece of code:

```
long SumIntegers(short a, short b)
{
  long   tempSum = 0;

  for (;a > 0; a--, b--)
    tempSum += a + b;
  return tempSum;
}
```

Simply changing the two input parameters from shorts to longs increases the speed of this loop by 30% on a PowerPC 601. (A 604, with its multiple integer execution units, can execute the extra sign-extensions for "free", but these extra instructions still consume valuable space within the instruction cache.)

In general, 32-bit integer quantities are more efficient than 16 or 8-bit, and unsigned are more efficient than signed.

Complex Arithmetic Operations

Most arithmetic and logical operations on PowerPC processors require a single cycle and have no added latency, i.e., the results of the operation can be used in the next cycle. There are a few complex arithmetic instructions, however, which take multiple cycles. Multiplies, for example, typically take between two and five cycles, and divides between 20 and 40 cycles. The actual cycle counts depend on the processor and the significant digits within the operands.

Multiplication and division are both very useful operations, so how can they be avoided in a time-critical loop? Most compilers attempt to optimize multiplication by a power of two by replacing the expensive operation with a single-cycle shift. There are occasions where multiplication by other constant values can be optimized in a similar manner. Take for example, the following calculation:

```
a = b * 40;
```

If we notice that 40 equals 32 plus 8, we can rewrite this as:

```
a = (b * 32) + (b * 8);
```

The compiler will now use shifts to optimize the code. The cycle count goes from five to three on a 601 and from four to two on a 604. (Unfortunately, it increases from two to three cycles on a 603 which has a relatively fast multiplier.) Note that we've increased the generated code size (one multiply becomes two shifts and an add).

Programmer Training

MACINTOSH SEMINARS & CONSULTING

Richey Software Training provides professional, *customized* programming seminars and industry consulting.

Rich content & hands-on lab exercises shorten your learning curve. On-site training is convenient – your team eliminates expensive travel costs and consuming down-time.

MAC SEMINARS

- C & C++
- OOP
- MacApp
- PowerPC
- AppleScript
- MPW
- System 7
- Debugging
- SourceBug

“*Professional training & consultancy that really hits the mark.*”

Call today to find out how Richey Software Training delivers professional seminars and consulting nationwide.

RICHEY SOFTWARE TRAINING

Training Mac Programmers Since 1986

707•869•2836

AppleLink: RICHEY.SOFT
INTERNET: 70413.2710@compuserve.com
P.O. BOX 1809, GUERNEVILLE, CA 95446-1809

© 1994 Richey Software Training. All trademarks or registered trademarks are the property of their respective owners. Specifications subject to change.

Multiplies are not much more expensive than a series of shifts and adds, so we cannot win much with this optimization, but division is so expensive, there is more room for improvement. Division by a power of two can also be changed into a shift as long as the dividend is unsigned. However, other constant values are more difficult. It is often necessary to give up some precision to construct a cheap replacement. Take, for example, the following calculations:

```
a = b / 13;
c = d / 40;
```

This piece of code can be replaced by:

```
// divide b by 12.800
a = (b / 16) + (b / 64);
// divide d by 39.385
c = (d / 64) + (d / 128) + (d / 512);
```

In this example, assuming *b* and *d* are unsigned, the compiler can use right shifts rather than expensive divisions, and the quotients will be accurate to within 1.5% (practically close enough for a Pentium!). How much speed has the change bought us? On a 601, the cycle count goes from 74 to 8; on a 603 from 76 to 8; on a 604 from 41 to 4. In short, this optimization is a huge win!

The large difference in performance between multiplication and division makes it tempting to try to replace one with the other. This is possible in some scenarios, as demonstrated in the following piece of code:

```
for (i=0; i<kMaxI; i++)
  dest[i] = src[i] * a / b;
```

Notice that values of *a* and *b* do not change throughout the loop. It would be much faster to calculate the fraction once and then use a single multiply within the loop. Unfortunately, calculation of the fraction *a/b* will result in a loss of significant digits. Luckily, if we can assume *a* and *b* are both 16 bits in length, we can use a shift operation to maintain more significant digits. The above code can be changed to:

```
unsigned long fract = (a << 16) / b;
for (i=0; i<kMaxI; i++)
  dest[i] = (src[i] * fract) >> 16;
```

The cycle count for the inner loop goes from 49 to 26 on a 601 and from 28 to 9 on a 604 – another huge performance win!

Floating Point Operations

One of the strengths of PowerPC processors is floating point performance. Unlike older architectures, PowerPC defines double precision floating point as mandatory for architectural compliance. This allows developers to use floating point without worrying about terrible performance on low-end platforms. If floating point makes sense in your program, by all means use it. Do not attempt to avoid it by using a complex fixed-point scheme as would have been suggested on older processors.

Floating point calculations are fast for two primary reasons. First, most PowerPC processors dedicate significant silicon area to floating point calculations. In fact, floating point multiplication and division are faster on most processors than the corresponding integer operations. Second, floating point is handled by a separate functional unit which operates, to a great extent, independently of the other functional units. While the other functional units are busy handling loads, stores, branches, and integer calculations, floating point operations can be interleaved with little extra cost.

Keep in mind that single-precision floating point (32-bit) operations are faster on most processors than double-precision (64-bit). Long double (128-bit) operations are not supported in hardware and must be emulated in software, often with a large performance hit. Be careful when selecting the level of precision your program requires.

Because the floating point unit is relatively independent of the other functional units, the PowerPC architecture includes no integer-to-floating-point conversion instructions. This means that frequent use of operations which intermix integer and floating point values can result in suboptimal performance. Both explicit and implicit type conversions should be kept to a minimum.

Branch Optimizations

Branches play a critical role in most programs, but they unfortunately don't perform any useful computation. For this reason, PowerPC processors attempt to fold branches out of the instruction stream to keep the other functional units fed without skipping a cycle. The majority of branches within a typical program are conditional, and good performance is dependent on the processor being able to predict whether or not each of these branches is going to be taken.

The 601 and 603 use simple rules to predict the direction of a branch. If a conditional branch is forward (i.e., has a positive offset), it is predicted to be not taken; backward branches are predicted taken. The architecture defines a special prediction override bit which can be set within the branch instruction to indicate that the normal prediction is probably not optimal. Unfortunately, compilers do not usually have enough information to accurately set these bits, and there is no way to tell the compiler which conditionals will typically be true. Most compilers simply leave these bits clear. This means that it is important to structure conditional statements in a way which will result in the best branch prediction. Since forward branches are predicted not taken, it is best to put the most frequently executed code in the *if* clause of an *if-else* statement.

The 604 uses a more sophisticated mechanism for branch prediction. It dynamically tracks branches under the assumption that a branch which was recently taken will probably be taken the next time it is encountered. There is very little that can be done from a programming perspective to increase the branch prediction success rate in this case.

Typical code often makes use of counters which are decremented each time through a loop. For this reason, many processor architectures provide instructions which perform a combined decrement/branch operation. This is true of both the 68K and PowerPC architectures. However, due to minor differences, high-level loops should be coded differently. Take the following piece of code written for the 68K:

```
// this loop initializes a block of memory to zero
for (x = blockLength - 1; x >= 0; x--)
  *blockPtr++ = 0;
```

The 68K *dbra* (decrement and branch always) instruction can be used at the end of this loop to decrement *x* and terminate the loop when the count becomes negative. The PowerPC *bdnz* (decrement and branch if not zero) instruction performs a similar function, but terminates the loop when the loop count reaches zero. Therefore, most compilers will generate better PowerPC code if the above example is changed to the following:

```
// this loop initializes a block of memory to zero
for (x = blockLength; x > 0; x--)
  *blockPtr++ = 0;
```

Subroutine Calls

Because of the PowerPC's ability to fold branches, subroutine inlining and loop unrolling is not as important as with other RISC architectures. In fact, unless a subroutine or loop is extremely simple, it is often best to avoid inlining and loop unrolling as these will only serve to increase code size (along with instruction cache misses, page faults, etc.).

Not all subroutine calls are this cheap, however. When the callee is implemented in a different shared library, an additional cost is incurred. These types of calls are called *cross-TOC* because the compiler-generated code performs a double indirection through the TOC to get to the subroutine. The code must first load a pointer to a *transition vector* which lives in the callee's data section. The transition vector contains a pointer to the actual subroutine code as well as a new TOC pointer for the callee's library.

At compile time, the compiler cannot usually determine whether a subroutine will be located within the same library as the caller, so it generates the following code sequence:

```
bl subroutine
nop
```

When all object files are linked to form a single library, the linker fills in the correct branch offset, and the *nop* instruction is left as is. If the linker does not find the subroutine within the library, it assumes it is an import symbol and generates a

Apprentice

Mac Source Code CD-ROM

New Release 2! Over 600 megabytes of up-to-date Mac-only source code and programmer utilities. Most of the source code is in CodeWarrior, Symantec, and MPW projects for C, C++, and Pascal. Includes complete working examples of applications, games, control panels, extensions, utilities, and much more! $35 including shipping. Add $5 for shipping outside of the U.S. and Canada. VISA, MC, American Express, and Discover gladly accepted.

Celestin Company, 1152 Hastings Ave, Port Townsend, WA 98368
800 835 5514 • 360 385 3767 • 360 385 3586 fax
Internet: celestin@olympus.net • CompuServe: 71630,650

Pascal → C++

Stuck with Pascal source code?

Want to move to C++?

OP2CPlus is a Macintosh tool for converting Object Pascal source code to C++ source code. It has already been used to translate hundreds of thousands of lines of Object Pascal to C++.

Ease your transition to OpenDoc or PowerPC

- ❑ Convert in days instead of months
- ❑ Generates C++ classes
- ❑ Works with MacApp 2 or 3
- ❑ Translates Think or MPW Pascal
- ❑ Full ANSI C source code included
- ❑ Just $895

Graphic Magic Inc, 180 Seventh Ave, Suite 201
Santa Cruz CA 95062 Tel (408) 464 1949
Fax (408) 464 0731 AppleLink GRAPHICMAGIC

small glue stub. This glue is responsible for saving the current TOC pointer, loading a transition vector pointer from the current TOC, loading a new TOC pointer and code address from the transition vector, and finally jumping to the subroutine. The *nop* instruction is overwritten with an instruction which restores the caller's TOC pointer after the return from the subroutine. The typical code sequence looks like the following:

```
    bl   subroutine_glue
    lwz  rTOC,20(SP)
```

subroutine_glue:

```
    lwz  r12,subroutineTVOffset(rTOC)
    stw  rTOC,20(SP)
    lwz  r0,0(r12)
    lwz  rTOC,4(r12)
    mtctr  r0
    bctr
```

The entire cross-TOC call sequence adds approximately ten cycles to the subroutine call, assuming there are no cache misses. Care should be taken to minimize the use of cross-TOC calls within time-critical code if possible.

When the linker determines that the subroutine is in the same library as the caller, the extra *nop* instruction is left unmodified. Luckily, the 603 and 604 fold *nops* out of the instruction stream with a zero-cycle penalty, but they still take up extra space. One way to help reduce the number of these unneeded instructions is to define functions as *static* where appropriate. When the key word *static* appears before a function prototype, the compiler can assume the routine will be implemented within the current file. The compiler can also assume a subroutine is local if its implementation appears before it is called. Take, for example, the following code sequence:

```
OSErr caller(void)  { return callee(void); }
OSErr callee(void)  {  return noErr; }
```

It is better to either reverse the implementations or declare the callee as static (if appropriate):

```
static OSErr callee(void);

OSErr callee(void)  {  return noErr; }
OSErr caller(void)  { return callee(void); }
```

Parameter Passing

The PowerPC runtime architecture makes use of the large on-chip register files wherever possible to reduce parameter passing overhead. Most CISC architectures, including the 68K, use stack-based calling conventions for most high-level languages. This means that passing small structures by reference is less costly than passing them directly, in which case the entire structure must be copied. This is not necessarily the case on the PowerPC. Take, for example the following routine:

```
long DotProduct1(Vector *v1, Vector *v2)
{
  return v1->x * v2->x + v1->y * v2->y;
}
```

Assuming the caller typically has the vector coordinates in registers, the subroutine can be optimized by passing the x and y coordinates of the two vectors by value.

```
long DotProduct2(long v1x, long v1y, long v2x, long v2y)
{
  return v1x * v2x + v1y * v2y;
}
```

Register-based parameter passing does have its limitations. Up to eight integer and thirteen floating point parameters can be passed in registers. Additional parameters must be spilled to the stack. Integer and floating point parameters overlap (e.g., if a 64-bit double is passed as the first parameter, it is assigned a floating point register, and the first two 32-bit general purpose registers are left unused). This means that long parameter lists containing both integer and floating point parameters will produce more optimal code if the floating point parameters are passed at the end of the parameter list.

Variable-length argument lists pose an interesting problem for register-based parameter passing conventions. The PowerPC runtime architecture calls for all variable argument lists to be spilled to the stack. Use of these routines should therefore be kept to a minimum in time-critical code.

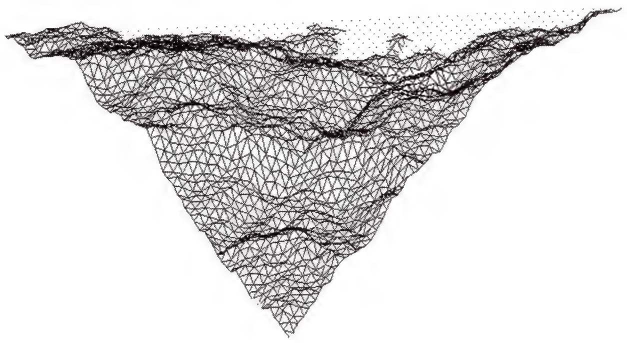
Mixed Mode

Subroutine calls which involve a Mixed Mode switch are much more expensive than simple or cross-TOC calls. Although Apple engineers have worked very hard to tune both Mixed Mode and the 68K emulator for optimal performance, the emulator should be avoided within time-critical code. Simply switching from native to emulated modes requires hundreds of cycles. Profilers and debuggers can help determine which system routines are implemented with native PowerPC code, which are "fat," and which are still emulated.

Mixed Mode calls which do not involve a mode switch (e.g., calls to toolbox routines which have already been ported) are much less costly but are still expensive relative to simple cross-TOC calls. Care should be taken to move these types of calls out of time-critical loops whenever possible.

Case Study: Fractal Contours

In an attempt to demonstrate some of the techniques discussed in this article, I went on a hunt for a piece of code which could benefit from tuning. My search ended on the Internet when I found the source for an old public domain Mac program called "Fractal Contours." This program was written in 1985 by Jim Cathey for the Mac Plus. The application generates impressive wire-framed mountainous landscapes. The source code for the program, along with the changes I made each step of the way, can be downloaded from any of MacTech's usual on-line sites.



Fractal 1

The version I found on the Internet no longer ran under System 7.5, so my first task was to modernize the code somewhat. I removed assumptions about the screen size, added a little bit of color and a menu option which toggles a "continuous redraw" mode. When this mode is enabled, fractal pictures are continuously calculated and displayed, and a frame rate (fractals per second) is computed. On a Power Mac 8100/80 using 8-bit video, I measured an initial frame rate of 0.59 fractals per second in emulation mode.

Fractal 2

Now that the program worked and I had defined a measure for performance, I could start tuning. The first step was to simply recompile the program for PowerPC. I chose Metrowerks' CodeWarrior C/C++ 1.2 for development. CodeWarrior's speedy turn-around times allowed me to try out many different performance optimizations and quickly measure their effects. With all of CodeWarrior's optimizations enabled, I immediately saw a 4.80x speed-up over the emulated version. I was up to 2.83 fractals per second.

Fractal 3

My first step in tuning was to find the hot spots within the program. I chose to use the adaptive sampling profiler built into the Macintosh Debugger. The first profile indicated that I was spending over 80% of the time in QuickDraw's StdLine function drawing lines to the screen. Because the screen is non-cacheable, QuickDraw is more efficient when working with RAM-based off-screen buffers. I changed the program to draw the fractal image to an off-screen buffer and then use CopyBits to copy it to the screen. The resulting frame rate was now 2.97 fractals per second, a rather disappointing 1.05x speedup over the previous version.

Fractal 4

A second profile showed that the program was still spending the majority of its time within the StdLine function. I noticed that all of the lines the program draws are typically very short (15 to 25 pixels), are always one pixel wide, never need clipping, and are always either black or blue. I decided that a custom line-drawing routine would help to remove the QuickDraw bottleneck. The result was a small increase in frame rate – now 3.47 fractals per second, a 1.17x speedup over the previous version.

Fractal 5

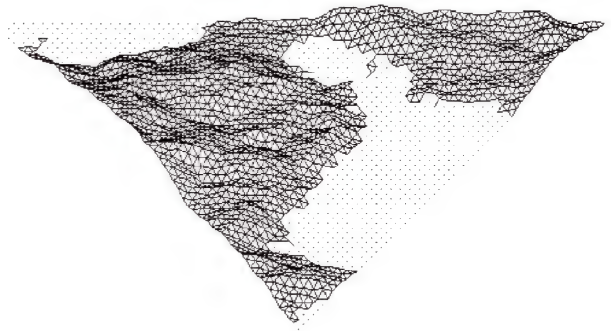
I was a little disappointed and confused as to why the custom line drawing routine did not improve performance further. I profiled the program once again. Indeed, I was no longer spending much time in StdLine, but I was spending a large amount of time in the routine GetPixBaseAddr and in the Mixed Mode Manager. GetPixBaseAddr is simply an accessor function and dereferences a PixMap to return its base address. A quick check revealed that the routine had unfortunately not been ported to native PowerPC. I was making a Mixed Mode switch each time I called this function. By removing the call from an inner loop and placing it in a less time-critical piece of code, I was able to increase performance substantially. I was now up to 6.69 fractals per second, an impressive 1.93x increase over the previous version. My custom line drawing routine was successful after all.

Fractal 6

I ran the profiler once again to find the remaining hot spots. My custom line drawing routine now came up on top. Using various techniques described above, I removed division and multiplication operations wherever possible and changed signed shorts to longs. I also added special cases for drawing horizontal and vertical lines, added static keywords to function prototypes, and passed points and vectors by their individual values rather than by reference. All of this work once again paid off. I was now seeing 10.92 fractals per second, a 1.63x increase over the previous version.

Fractal 7

One last profile revealed that I was still spending a large amount of time in the line drawing routines. Confident that I had already tuned these as much as I could, I concentrated on other hot spots within the code. I noticed quite a bit of time was being spent scaling and transforming the points so the two-dimensional array of *xy* coordinates could be drawn in a three-dimensional view. The original program used a *cordic* algorithm to iteratively approximate sines and cosines. I had added another scaling operation to magnify the image from its original size – the size of a Mac Plus screen – to the size of the window. I found that an algorithmic change was needed, so I devised a simple linear algebra technique to scale, rotate, and skew the image in one step. The resulting image looks slightly different from the original, but the effects are still impressive – and so was the speed improvement. The final frame rate was now 27.00 fractals per second. This is 9.54x faster than the first native version and 45.76x faster than the emulated version! The tuning efforts definitely paid off.

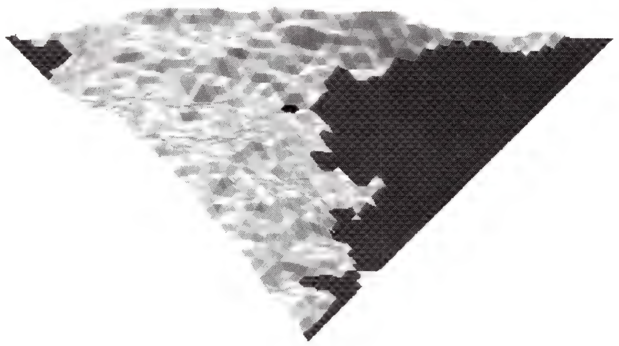


Fractal 8

Now that I could achieve almost thirty frames per second, I decided to add some more interesting features to the program. I first incorporated a "morphing" feature which draws intermediate fractals so it appears that each one morphs into the next. Next, a friend contributed some triangle blitting code he had written, and we added a surface mapping feature. The result is quite entertaining. Fully shaded mountains rise and fall in an ever changing pattern. None of this animation would have been possible if I had been stuck with the 2.8 frames per second I achieved by simply porting the program native.

Appeal for Innovation

As the fractal example shows, performance tuning doesn't just mean a faster application. It can enable a program to do things that wouldn't otherwise be possible.



Applications which are noticeably lethargic are no fun to use. Really fast programs which simply do their job faster are less annoying, but still often boring. Programs which make use performance in innovative ways, on the other hand, will always be in demand. PowerPC offers Mac developers an excellent opportunity to deliver these types of applications. When you're planning the next version of your application, keep in mind the performance tuning tips discussed in this article – and use your imagination to add the features that will make your program really stand out.

For More Information, Refer To:

Inside Macintosh, PowerPC System Software, 1994. Addison-Wesley Publishing Co.

Inside Macintosh, PowerPC Numerics, 1994. Addison-Wesley Publishing Co.

Macintosh Debugger Reference, 1994. Apple Computer, Inc.

Parent, Sean. "Living in An Exceptional World", **develop**, Issue 11, August 1992. Apple Computer, Inc.

PowerPC 601 User's Manual. 1993, Motorola, Inc.

PowerPC 603 User's Manual. 1994, Motorola, Inc.

PowerPC 604 User's Manual. 1994, Motorola, Inc.

UNIFORM RESOURCE LOCATORS

By Scott T Boyd and John Kawakami

Save yourself from overexerting your digits, and find this page online at **http://www.class.com/MacTech/URLs.html**

What's New and Interesting?

Dave Winer and the wizards at UserLand have released a radically simple WWW publishing tool called AutoWeb. You give it TEXT or Word files, and AutoWeb produces working web hypertexts, with links and graphics. **http://garnet.msen.com:80/~dwiner/**

CI Labs has improved their Web site, and it now includes some fine reading material about OpenDoc. **http://www.cilabs.org/**

Everyone knows about the very hot-yet-hoggy NetScape; MacWeb is a much smaller web browser. **http://galaxy.einet.net/EINet/MacWeb/MacWebHome.html**

Fun and Games

Networked games have escaped the confines of the office and lab clusters and invaded the Internet. Can we say "business opportunity?"

While NetTrek is the granddaddy of Mac net games, Bolo is no newbie, either. *[It's absolutely addictive, even if you play as badly as I do- jtk]* **http://bolo.ncsa.uiuc.edu/**

Outland is a Macintosh-specific commercial service featuring a pay-to-play version of Spaceward Ho!, among others. They use special client software you can get from: **ftp://ftp.outland.com/**

Internetrivia

Once you've hooked yourself up to receive information from the Net, the next logical step is to get your computer on the Net. InterNIC, an umbrella agency for the three network information centers, runs the definitive Net information server. **http://www.internic.net/**

If you ever wondered what you could be doing if you were writing papers instead of programs, take a quick trip to The English Server. It's one of the most interesting libraries of articles, books, and term papers on the WWW. Rumor has it that this site is hosted on a Macintosh. **http://english.hss.cmu.edu/**

Thanks this month to Bob Hettinga and Jim Straus.

Interesting Macintosh Places

New URLs and interesting places are in boldface.

Ada	**ftp://ftp.seas.gwu.edu/pub/ada**
Alpha	ftp://cs.rice.edu/public/Alpha/
Apple	ftp://ftp.info.apple.com
see also	http://www.info.apple.com/dev/dts.html
see also	http://www.austin.apple.com
Applescript	ftp://gaea.kgs.ukans.edu/applescript
BBEdit	ftp://ftp.netcom.com/pub/bb/bbsw
Bolo	**http://bolo.ncsa.uiuc.edu/**
Celestin	ftp://ftp.teleport.com/vendors/cci/apprentice
CIL	**http://www.cilabs.org/**
CU-SeeMe	http://www.indstate.edu/CU-SeeMe/index.html
Dylan	http://www.cambridge.apple.com/
see also	ftp://cambridge.apple.com/pub/dylan
see also	http://legend.gwydion.cs.cmu.edu:8001/dylan
see also	news://comp.lang.dylan
Get1Resource	http://www.asel.udel.edu/~haynes/g1r.html
Info-Mac searcher	**http://www.mid.net/INFO-MAC**
Internet Config	ftp://ftp.share.com/internet-configuration/
InterNIC	**http://www.internic.net/**
ISDN page	**http://alumni.caltech.edu/~dank/isdn/**
Kaleida	**http://www.kaleida.com/**
Robert Lentz	http://www.astro.nwu.edu/lentz/mac/programming/home-prog.html
Peter Lewis	ftp://amug.org/pub/peterlewis
Lisp	http://www.cambridge.apple.com/mcl/mcl.html
see also	http://www.cs.rochester.edu/u/miller/alu.html
and	**http://www.digitool.com/**
The Macintosh Vendor Directory	http://rever.nmsu.edu/~elharo/faq/vendor.html
Hacking MacHTTP	http://www.uwtc.washington.edu/Computing/WWW/ExtendingMacHTTP.html
Mac Netswitch	http://www.nd.edu/~dwalton1/
MacNosy	ftp://ftp.netcom.com/pub/ma/macnosy
MacTech	**ftp://ftp.netcom.com/pub/xp/xplain**
MacWeb	**http;//galaxy.einet.net/EINet/MacWeb/MacWebHome.html**
Metrowerks	http://www.iquest.com/~fairgate/cw/cw.html
Matthias Neeracher	http://err.ethz.ch/members/neeri.html
nick.c	http://www.pitt.edu/~nick/
OpenDoc	**ftp://cilabs.org/pub/cilabs**
Outland	**ftp://ftp.outland.com/**
François Pottier	http://acacia.ens.fr:8080/home/pottier/index.html
see also	news://comp.sys.mac.digest
Digests archive	**ftp://ftp.dartmouth.edu/pub/csmp-digest**
Jon Pugh	**ftp://ftp.netcom.com/pub/jo/jonpugh/homepage.html**
Paul Robichaux	**http://www.iquest.com/~fairgate**
QKS/Smalltalk	http://www.qks.com
QuickCam	http://www.engin.umich.edu/~friscolr/QuickCamtm/readme.html
Eric Scouten (TCP)	**http://tampico.cso.uiuc.edu/~scouten/**
Symantec	ftp://devtools.symantec.com/Macintosh/Updaters/DevTools
Taligent	**http://www.taligent.com/**
TCL stuff	ftp://daemon.ncsa.uiuc.edu/TCL
TidBITS	http://www.dartmouth.edu/pages/TidBITS/TidBITS.html
see also	news://comp.sys.mac.digest
Time Tracker	ftp://ftp.maui.com/pub/mauisw/
Dave Winer	**http://garnet.msen.com:80/~dwiner/**

MacRegistry™

Developer Job Opportunities

If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

800-231-5920 | das@spi.com | Fax 713-496-0373

Scientific Placement, Inc.

MT, Box 19949, Houston, TX 77224 713-496-6100 Fax: 713-496-0373
MT, Box 71, San Ramon, CA 94583 510-733-6168 Fax: 510-733-6057
MT, Box 4270, Johnson City, TN 37602 615-854-9444 Fax: 615-854-9454
MT, Kenmore Station, Box 15225, Boston, MA 02215 617-424-8372 Fax: 617-424-7158
AppleLink: D1580; Compuserve: 71250,3001, eWorld: spi, AOL: davesmall

Cash redemption value 1/100 - cent. Limit 1 per customer

For MacApp Programmers

Section 4.3 Company Policy -Expenses: On any night where 3 or more people are concurrently working overtime, then any purchase of pizza will be reimbursed.

MacXperts has openings for experienced C++ and MacApp programmers. If you have what it takes, and the desire to achieve, call Kendall Tyler at MacXperts.

Voice: 800-356-8040 Fax: 804-358-3847
Internet: xperts@infi.net
AppleLink: xperts AOL: MacXperts

MAC PROFESSIONALS

Manpower Technical, a leader in the Technical Services industry, has current and upcoming contract openings for experienced MAC personnel with the following expertise: Software Development, Network Administrators, Desk Top Publishing, Help Desk, Applications Support, Technical Writers/Editors.

Positions are in California and throughout North America.

Interested persons are encouraged to send their resume to:

Manpower Technical
ATTN: Dept. MT
P.O. Box 2053
Milwaukee, WI 53201
(800) 558-6992
Fax # (414) 332-0378

Do you have used equipment to sell?

Or do you want to purchase used equipment?

Advertise it in the Classified Section of MacTech Magazine. Call Ruth Subrin at 310/575-4343 to find out how to place your ad.

CHICAGO AREA

PARALLEL SOFTWARE

Parallel Software is looking for software engineers who are serious about component based **OPENDOC** object development. We are interested in Mac or Windows people with the following experience:

"C" or "C++", Pascal, AppWare Newton, NDK, or Winpad MacApp, Code Warrior, MacToolbox Oracle, Sybase, or SQL Databases

Please write or phone:

Mr. John D. McMahon
Parallel Software, Inc.
608 S. Washington, Ste. 101
Naperville, Illinois, 60540
Tel (708)369-0100
Fax: (708)355-7870
psgmcmahon@aol.com.

"YOUR RECRUITMENT AD HERE"

Make direct contact with over 30,000 Macintosh programmers and developers by placing a low cost classified recruitment ad in MacTech Magazine's career opportunites section, *The Classifieds.* Whether pub set or your display ad, this is *the* direct approach to Macintosh programmers and developers.

For more information about placing your ad, call Ruth Subrin at 310/575-4343.

Not All Engineers Consider Multimedia A Serious Opportunity To Make Medical Breakthroughs.

But Ours Do.

"Multimedia" is most commonly a buzzword for a way to convey information or entertainment. But at the AEGIS™ Division of Acuson in Ann Arbor, we see it as something more — a new tool to help medical professionals detect health problems, preventing further injury or even loss of life. Multimedia capabilities now allow us to combine multiple sources of information into a single high quality presentation of moving images and audio.

By using their talents to create highly-advanced, yet highly usable, ultrasound image management software, the engineers at Acuson gain more than just professional rewards. They gain the satisfaction of developing technology with a greater purpose. Consider a career that offers the very best in technology — and in all that it inspires.

Software Engineer (Macintosh)

Working with other team members, you'll be responsible for adding features and improvements to Acuson's AEGIS product, a medical image management system running on a Macintosh network. In addition to 2+ years' Macintosh programming experience in C or C++, you'll need a BSCS/EE, or the equivalent, with a Masters Degree preferred. Dept. PM/5-012A.

Database Software Engineer

You'll be responsible for implementing a significant portion of the client/server software for the AEGIS cardiology product. This position requires a BS in Computer Science/Software Engineering or the equivalent, and 2+ years' experience developing database software in a Macintosh or Windows environment. Experience with C is required, and C++/object-oriented design/programming expertise is preferred. Experience with products in medical imaging is also a plus. Excellent communication skills are required. Dept. PM/AA1.

Senior Software Test Engineer

In this key spot, you'll take the lead on the team testing the AEGIS cardiology product. Working with, and giving guidance to, other software engineers, you'll learn the product, assess the stability of the software, and focus the testing efforts on critical portions. Our successful applicant should have completed at least one Macintosh/Windows product development cycle, and should be able to write scripts to automate the testing process. Our ideal candidate will be able to participate in design reviews to help design testability. In addition to your BS in Computer Science/Software Engineering, or equivalent experience, you'll need excellent verbal and written communications skills, and 2+ years' experience of software testing in a Macintosh or Windows environment. Experience with QuickTime-based products is preferred. Dept. PM/AA2.

Software Test Engineer

Drawing on your background in Macintosh and Windows, you'll be responsible for testing the components of the AEGIS cardiology product. Some software test experience will be helpful, but not required. You'll need a BS in a scientific discipline, or the equivalent, good Macintosh/Window computer proficiency, and excellent written/oral communication skills. Dept. PM/AA3.

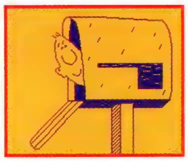
Macintosh User Interface Software Engineer

Working with our marketing and development groups and end-users, you'll implement and improve our user interfaces. You should have a BSCS, with a Masters Degree preferred. You'll also need 4 years' experience developing user interfaces for Macintosh or Windows, as well as a background in C programming. Experience with C++, and object-oriented design and programming is a plus. This position requires good written/oral communication skills, and course work or experience with imaging will be helpful. Dept. PM/5-022A.

The benefits of working at Acuson in Ann Arbor are many. Please send your resume, indicating appropriate Dept. to: Acuson, 1220 Charleston Road, P.O. Box 7393, Mountain View, CA 94039-7393; FAX (415) 961-4726. EOE.

acuson
CORPORATION

TECHNOLOGY WITH A GREATER PURPOSE.



By Scott T Boyd, Editor

Jasik Has A Better Bus Error

Steve Jasik wrote, in light of the attention we've been giving to the need to use strong debugging tools, to remind us that his Debugger has been putting a "garbage" value in location zero since its inception, and you get the message if you trip on it.

He also wanted to remind us that it has a version of Trap Discipline similar to the one implemented in TMON, but it also does the Color QD Traps. His version of Discipline will catch double disposes, etc. He says it's supported (and documented?).

– *Ed stb*

Stenger Goes For Two Month's In A Row

I found a couple of cases where Bob Boonstra's solution to the Rubik's Cube challenge (*MacTech*, February 1995, pp. 49 ff.) goes into a loop.

(1) If the cube is already solved, the code correctly detects that no moves are necessary, but the playback loop in SolveRubiksCube doesn't handle this empty case correctly (the loop is coded to "execute at least once").

(2) The illegally-colored cube with solid faces 0, 3, 4, 2, 1, 5 is not caught by LegalCube, and then generates no moves (I don't know why this is). It then fails as in item (1).
The code works correctly on all the "normal" cases I tried, so it meets the spirit of the Challenge and should still be the winner.

– *Allen Stenger, 70401.1171@compuserve.com*

Prograph – Where's It Been Hiding?

I read Kurt Schmucker's article on Prograph CPX, and was amazed that I had never heard of this product. I am a programmer for a consulting company here in Milwaukee. After reading that article, I went out and bought Prograph CPX immediately. I want to see more articles about Prograph – I think it's a product that has a great future.

– *minton brooks, MBAssoc@aol.com*

Re: A Quick Trip Into the Depth

In his article in the February issue, "A Quick Trip Into the Depths", Steve Jasik writes, "The crux of the problem was that, despite what *Inside Macintosh* or today's equivalent of it says, the Resource Manager doesn't always return a non-zero value of ResErr when it doesn't find the requested resource."

This assertion is incorrect. *Inside Macintosh* does specifically warn you about this problem. *Inside Macintosh : More Macintosh Toolbox* on page I-51 in describing ResError function has this to say under the caption IMPORTANT, "In certain cases, the ResError function returns noErr even though a Resource Manager routine was unable to perform the requested operation. See the individual routine descriptions for details about the circumstances under which this happens".

Then if we turn to page I-73 of the same volume, we can read that, "If you call GetResource with a resource type that can't be found in any of the resource maps of the open resource forks, the function returns NIL, but ResError returns the result code noErr. You should always check that the value of the returned handle is not NIL."

– *Vladimir Potap'yev, VolodyaP@aol.com*

We Forgot To Say Thanks!

Chris De Salvo contributed Control Strip Tester – an application shell to help you test out control strips – to go along with Mike Blackwell's "Writing Control Strip Modules" article in our December issue. We didn't have time to get a mention of it into the issue, but we did manage to get it onto our source disk and our online sites. You can find it at:
ftp://ftp.netcom.com/pub/xp/xplain/Source/10.12/ControlStripExtras.sit.bin
and our other online sites, and you can e-mail him at phixus@netcom.com.

Thanks, Chris!

– *Ed stb*

Dilbert® *by Scott Adams*

CHAPTER IV. "TIME MANAGEMENT"
"ALWAYS POSTPONE MEETINGS WITH TIME-WASTING MORONS."
HOW DO YOU DO THAT?
CAN I GET BACK TO YOU ON THAT?
S.Adams
ScottAdams@aol.com
© 1989 United Feature Syndicate, Inc.
5-10

reprinted by permission of UFS, Inc.

NEWSBITS

By Scott T Boyd, Editor

New NeoLogic

NeoLogic Systems announced the release of NeoAccess 3.1, an update of NeoLogic's cross-platform object database engine. This latest release includes full support for all Unix platforms in addition to the Windows, Macintosh and DOS support provided in previous releases.

NeoAccess 3.1's new features include full support for all Unix platforms, enhancements to the metaclass facility (including the ability to add members), improved support for dynamically adding and removing indices from a class, improvements in consolidated index support, and even more powerful object iterator support.

The NeoAccess Developer's Toolkit sells for $749 per developer (no runtime licensing fees). It includes full source code, sample applications, 450+ pages of docs, and 30 days of online technical support.

Credit card orders (800) 799-4737. Save an additional $100 if you purchase the product off the Internet (ftp://ftp.neologic.com/users/neologic/neoaccess.sea.hqx), Metrowerks' CodeWarrior or Symantec's Developers Advantage CD-ROMs.

NeoLogic Systems 1450 Fourth St., Suite 12 Berkeley, CA 94710
(510) 524-5897 voice, (510) 524-4501 fax, neologic@neologic.com,
http://www.neologic.com/~neologic/

Version Control

Barking Dog Software Co. has released "Version Control" for both PowerMac and 68K Macintosh systems. "Version Control" is a stand-alone program for source code and document revision control and release management. It allows users to recover past versions of single files or easily extract entire releases composed of hundreds of files using a snapshot feature. The software allows easy tracking of who made file changes, when the changes were made, and why the changes were made. A complete audit trail of the file history is available for bug tracking, specific version source reconstruction and change management. "Version Control" utilizes Drag & Drop for check-ins.

A single user license of "Version Control" is $199, 2 users $249, 3-5 users $349, and 6-10 users $499.

Barking Dog Software Co., 4822 Santa Monica Ave., Suite 179, San Diego, CA 92107. (619) 222-8361, calltree@aol.com.

Mercutio – Powerful, Cheap MDEF

Digital Alchemy announces the release of version 1.2 of the Mercutio, available from any info-mac site, including ftp://ftp.hawaii.edu//mirrors/info-mac/dev/mercutio-mdef-12.hqx

The Mercutio MDEF allows developers to easily extend the power of their application menus. Menus may have multiple-modifier key-equivalents (e.g. shift-command-C), custom icons, item callbacks, and other goodies. Mercutio includes System 7's Balloon Help and True Gray, Color menus, Small and large icons, SICNs in hierarchical menu items, and 99% compatible with the standard MDEF.

Major feature additions and changes since 1.1.5 are:

- Support for non-printing keys (function keys, page up/down, arrow keys, etc)
- Support for icon suites
- Support for all 4 modifier keys (command, shift, option, control)
- Removed command-key requirement
- User-definable style mapping: developer determines which style bits are used as feature flags
- Dynamic items: items whose contents change depending on what modifier keys are being held down (e.g. when the option key is held down, Save becomes Save All).
- Application callback routine to let you decide on the fly what the item contents should be.

The Mercutio MDEF works with System 6.0.4 or later, with or without Color QuickDraw. Mercutio is fully compatible with System 7, and supports all the features of the System MDEF. Integrating the MDEF into your program shouldn't take more than 15 minutes.

The documentation for Mercutio is in Adobe's Acrobat format. To obtain a free Acrobat reader application, FTP from: ftp://ftp.adobe.com/pub/adobe/Applications/Acrobat/Macintosh/AcroRead.sea.hqx

Developers may use Mercutio free of charge as long as they give credit in the About box and documentation, and send a copy of the final program (including future upgrades for as long as they use the MDEF). Other licensing terms are available.

ShareWare Authors Take Note

CompuServe offers a Shareware Registration service (GO SWREG) as part of its basic services. CompuServe provides a convenient, secure distribution and support environment for both shareware developers and users. Nearly 4,000 authors participate in the registration service.

CompuServe members simply look up the software online that they want to register and follow on-screen instructions. The cost of the product appears on their next CompuServe bill. CompuServe offers developers a convenient way to track registrations. The developer interface permits varied shipping & handling pricing based on the user's location. Developers can edit and update their product descriptions as needed.

CompuServe members often help with the development process as beta testers. The Shareware Beta Forum (GO SWBETA) allows shareware authors to test applications prior to release. The Association of Shareware Professionals (ASP) manages a forum on CompuServe. A number of authors use the shareware registration service to get access to sales they might not have made otherwise.

Newsbits continued from page 85

MacTech Magazine Seen On the Web

Get1Resource, Carl Haynes' Web 'zine for Mac programmers, included the following tidbit, "The January issue of MacTech Magazine included CDs to encourage developing of parts for both OpenDoc from CIL, and OLE from Microsoft. I poked around a little with the OpenDoc stuff, and like probably 90% of the readers threw my OLE disk into my pile of CDs that I'll never look at again." http://www.asel.udel.edu/~haynes/g1r.html

EFF Fights for Cryptography Rights

In a move aimed at expanding the growth and spread of privacy and security technologies, the Electronic Frontier Foundation is sponsoring a federal lawsuit seeking to bar the government from restricting publication of cryptographic documents and software. EFF argues that the export-control laws, both on their face and as applied to users of cryptographic materials, are unconstitutional.

EFF believes that cryptography is central to the preservation of privacy and security in an increasingly computerized and networked world. Many of the privacy and security violations alleged in the Kevin Mitnick case, such as the theft of credit card numbers, the reading of other people's electronic mail, and the hijacking of other peoples' computer accounts, could have been prevented by widespread deployment of this technology. The U.S. government has opposed such deployment, fearing that its citizens will be private and secure from the government as well as from other vandals.

The plaintiff in the suit is a graduate student in the Department of Mathematics at the University of California at Berkeley named Daniel J Bernstein. Bernstein developed an encryption equation, or algorithm, and wishes to publish the algorithm, a mathematical paper that describes and explains the algorithm, and a computer program that runs the algorithm. Bernstein also wishes to discuss these items at mathematical conferences and other open, public meetings.

The problem is that the government currently treats cryptographic software as if it were a physical weapon and highly regulates its dissemination. Any individual or company who wants to export such software – or to publish on the Internet any "technical data" such as papers describing encryption software or algorithms – must first obtain a license from the State Department. Under the terms of this license, each recipient of the licensed software or information must be tracked and reported to the government. Penalties can be pretty stiff – ten years in jail, a million dollar criminal fine, plus civil fines. This legal scheme effectively prevents individuals from engaging in otherwise legal communications about encryption.

The lawsuit challenges the export-control scheme as an "impermissible prior restraint on speech, in violation of the First Amendment." Software and its associated documentation, the plaintiff contends, are published, not manufactured; they are Constitutionally protected works of human-to-human communication, like a movie, a book, or a telephone conversation. These communications cannot be suppressed by the government except under very narrow conditions – conditions that are not met by the vague and overbroad export-control laws. In denying people the right to publish such information freely, these laws, regulations, and procedures unconstitutionally abridge the right to speak, to publish, to associate with others, and to engage in academic inquiry and study. They also have the effect of restricting the availability of a means for individuals to protect their privacy, which is also a Constitutionally protected interest.

More specifically, the current export control process:

- allows bureaucrats to restrict publication without ever going to court;
- provides too few procedural safeguards for First Amendment rights;
- requires publishers to register with the government, creating in effect a "licensed press";
- disallows general publication by requiring recipients to be individually identified;
- is sufficiently vague that ordinary people cannot know what conduct is allowed and what conduct is prohibited;
- is overbroad because it prohibits conduct that is clearly protected (such as speaking to foreigners within the US);
- is applied overbroadly, by prohibiting export of software that contains no cryptography, on the theory that cryptography could be added to it later;
- egregiously violates the First Amendment by prohibiting private speech on cryptography because the government wishes its own opinions on cryptography to guide the public instead; and
- exceeds the authority granted by Congress in the export control laws, as well as exceeding the authority granted by the Constitution.

If this suit is successful in its challenge of the export-control laws, it will clear the way for cryptographic software to be treated like any other kind of software. This will allow companies to build high-quality security and privacy protection into their operating systems. It will also allow computer and network users, including those who use the Internet, much more freedom to build and exchange their own solutions to these problems, such as the freely available PGP encryption program. And it will enable the next generation of Internet protocols to come with built-in cryptographic security and privacy, replacing a sagging part of today's Internet infrastructure.

Civil Action No. C95-0582-MHP was filed in Federal District Court for the Northern District of California. EFF anticipates that the case will take several years to win. If the past is any guide, the government will use every trick and every procedural delaying tactic available to avoid having a court look at the real issues. Nevertheless, EFF remains firmly committed to this long term project. Once a court examines the issues on the merits, the government will be shown to be violating the Constitution, and that its attempts to restrict both freedom of speech and privacy will be shown to have no place in an open society.

Full text of the lawsuit and other paperwork filed in the case is available from the EFF's online archives. The exhibits which contain cryptographic information are not available online, because making them publicly available on the Internet could be considered an illegal export until the law is struck down. See http://www.eff.org/pub/EFF/Policy/Crypto/ITAR_export/Bernstein_case/

Tip of the Month continued from page 96

```
            TEHandle textBlock = NULL;

            SetRect (&viewRect, 15, 15, 165, 31);
            SetRect (&destRect, 15, 15, 165, 31);
            textBlock = TENew (&destRect, &viewRect);

            if (textBlock)
              {
               TECustomHook (intDrawHook,
                 (ProcPtr)PasswordDrawProc, textBlock);
               TECustomHook (intWidthHook,
                 (ProcPtr)PasswordMeasureProc, textBlock);
              }
            return textBlock;
      }
#ifdef __cplusplus
}
#endif
```

– Eric Rosé, rose@nomos.com

SyncWait No More!

Sometimes when you're AppleTalking or modeming and something goes wrong (like you switch the modem off while data is being sent to it), the computer will hang. The mouse will still move, but clicking will have no effect. Here's the solution: Drop into MacsBug. You should see the routine name "_vSyncWait" plus something as the current location. If you don't, you probably hit the system while it was doing something else. Hit Command-G to get back out of MacsBug, and try again. After a few tries you should find _vSyncWait.

_vSyncWait is the routine that the system uses to wait for some input from a serial port. If you can read assembly code, you'll see that it's pretty simple. Here's the dump of the significant part:

```
+0000  4080BB8C   MOVE.W $0010(A0),D0            | 3028 0010
+0004  4080BB90   BGT.S  _vSyncWait    ; 4080BB8C | 6EFA
```

Register A0 is pointing to a system data structure, in which a word will be cleared when the awaited input arrives. The MOVE.W instruction grabs this word and puts it into register D0. The BGT.S instruction then Branches back to the MOVE.W if the byte it just fetched is Greater Than zero (hence BGT). So it happens that this byte is never going to arrive for whatever reason, but the computer is going to wait for eternity. The secret to fixing this is to use Command-T to go step along until the MOVE.W instruction is displayed as the current instruction. Now use the "sw" command to set "@(a0+10)" to zero: `sw @(a0+10) 0`

Then hit Command-T twice more. The MOVE.W instruction will take the zero you just set into memory and put it in D0, so the D0 display on the right of the screen should have its right four digits all zeros. Then when you execute the BGT.S instruction, it should not go back to the MOVE.W since zero is not greater than zero.

Hit Command-G to go. If this was the only byte the software was waiting for, then it should continue running, although it may go a little crazy because it's been suddenly disconnected from whatever peripheral it was talking to. Quit the program, fix your hardware, and try again.

– Macneil Shonle and Dustin Mitchell
MacneilS@aol.com and DustyM2@aol.com

BBEdit 3.1

Not Your Father's Text Editor.

We've taken the accelerated for PowerPC™ text editor that you have come to rely on, put it on CD-ROM and added new features that we know your father would have wanted.

- Integrated support for Metrowerks CodeWarrior.
- Automatic text wrapping without those grungy carriage returns.
- The integrated PopupFuncs support now recognizes Rez, Fortran and 68K assembler source files.
- Electronic documentation with interactive browser, so the information you need is always at your fingertips.
- Tools from Bare Bones Software's freeware collection.
- BBEdit freeware and shareware extensions.
- The BBEdit Extension Developer's Kit.

Take some really cool software for a spin.
On top of all of that, the CD includes demos, updates, special offers and information from industry leaders, including: Aladdin Systems, Language Systems, MacTech Magazine, Mathemæsthetics, MindVision Software, Onyx Technology and Symantec.

How do I get mine?
Look in the Mail Order Store section of this issue for info about ordering BBEdit 3.1 or give us a call.

For a free demo disk, send us e-mail:
Internet: bbsw@netcom.com
CompuServe: 73051,3255
Applelink: BARE.BONES
E-World: BareBones

BBEdit 3.1, It Doesn't Suck.

Bare Bones Software, Inc.

Accelerated for Power Macintosh

Mac™OS

We also have "It doesn't suck" t-shirts that don't either. Call us to order. © 1994 Bare Bones Software, Inc. P.O. Box 108 Bedford, MA 01760 voice: 508.651.3561 fax: 508.651.7584. Our lawyers insist we say: BBEdit, PopupFuncs and our spiffy logo are trademarks of Bare Bones Software, Inc. Mac and the Mac OS logo are trademarks of Apple Computer, Inc., used under license. PowerPC is a trademark of International Business Machines Corporation. All other trademarks and registered trademarks are properties of their respective holders.

If you have a CD-ROM drive, then *you've gotta have* every article published in the first 9 years of MacTech Magazine!!

FREE UPGRADE TO VOL 1-10!*

... now in THINK Reference format!

Availabl Now!!

MacTech CD-ROM, Volumes 1-9:

Over 1100 Articles.
All 103 Issues, including all of 1993.
All the Source Code.
THINK Reference 2.0.
Working Applications.
Full Documentation.
Demos for Developers.
And More!!

"When I designed THINK Reference, I envisioned endless databases at my fingertips. MacTech has doubled the information that is just a mouse click away."
– *Darrell LeBlanc, Formerly of Symantec, Author, THINK Reference 2.0*

"The CD strikes me as an impressive and very useful resource. The only disadvantage I've found is that each answer the databases yield exposes me to so many more issues, that I find myself exploring the articles for the sheer wonder of it, and thus putting off the real coding I should be doing. :-)"
– *Nicholas De Mello MacTech CD Beta Tester*

MacTech MAGAZINE™ Formerly MacTutor
FOR MACINTOSH PROGRAMMERS & DEVELOPERS

Voice: 310/575-4343 • Fax: 310/575-0925
AppleLink: MT.CUSTSVC
CompuServe: 71333,1063
Internet: custservice@xplain.com
America Online & GEnie: MACTECHMAG

✓ **45 MB of MacTech Magazine articles, Volumes 1-9**
Every article, 1100+ of them, from all 103 issues of MacTech Magazine printed from 1984 through 1993. Articles ranging from Assembly to BASIC, C to Pascal, Forth to FORTRAN and more. *And the articles are in THINK Reference!*

✓ **Hyperlinks to *Inside Macintosh* Databases of THINK Reference**
The articles have hyperlinks to relevant portions of the *Inside Macintosh* databases of THINK Reference. For example, if you are looking for information on Aliases, look at the MacTech articles on Aliases and use the hyperlinks to jump to the *Inside Macintosh* entry for the Alias Manager. And now, with the articles in THINK Reference, you can do free-text searches 8-10 times faster than you could previously with On Location™ 2.0.

✓ **100 MB of MacTech Magazine source code examples, samples, and utilities.**
These are the files that go with the magazine – the code that the articles are talking about. Use them in your own applications, with no royalties!

✓ **A fully-capable version of THINK Reference 2.0.3**
Including *Inside Macintosh* Volumes 1-6 (7 MB).

✓ **Sprocket – MacTech's Tiny Framework.**
Build your simple application using a lean, mean application framework. Experiment with new code without having to build a whole application around it!

✓ **80 MB of FrameWorks, MacApp®, MADA and SFA articles, files and source code.**
The most complete set of FrameWorks archives known.

✓ **Apple APIs, Utilities, and SDKs**
Including: Universal Header Files, Discipline, Macintosh Drag and Drop SDK, MacsBug, Telephone Manager SDK, Thread Manager SDK, TrueEdit 1.8 and and more.

✓ **Ariel Publishing's *Inside BASIC* on disk**
...and other related BASIC programming information and tools.

✓ **75 MB of Special Demos relevant for developers**

***FREE!* THINK™ Reference**
Symantec's THINK Reference 2.0. Complete on-line guide to Inside Macintosh, Vol. I-VI with cross referenced index, detailed information of each function, procedure and detail needed when programming the Macintosh.
SYMANTEC.™

MacTech CD-ROM, Volumes 1-9:
$199 plus shipping and handling. $69 plus shipping and handling for upgrades from any previous version of the CD

* All purchasers of the 1-9 CD will receive a FREE upgrade to the next version of the CD when in becomes available.

MACTECH MAGAZINE PRODUCTS & ORDER INFORMATION

E-mail, Fax, write, or call us. You may use your VISA, MasterCard or American Express; or you may send check or money order (in US funds only): MacTech Magazine, P.O. Box 250055, Los Angeles, CA 90025-9555. Voice: 310/575-4343 • Fax: 310/575-0925

If you are an e-mail user, you can place orders or contact customer service at:
- **AppleLink:** MT.CUSTSVC
- **CompuServe:** 71333,1063
- **Internet:** custservice@xplain.com
- **America Online:** MT CUSTSVC
- **GEnie:** MACTECHMAG

SUBSCRIPTIONS

US Magazine: $47 for 12 issues
Canada: $59 for 12 issues
International: $97 for 12 issues
Domestic source code disk: $77 for 12 issues
Int'l source code disk: $97 for 12 issues

CD-ROM

MacTech CD-ROM, Volumes I-IX: Includes over 1100 articles from all 103 issues (1984-1993) of MacTech Magazine (formerly MacTutor). All article text and source code. Now in THINK Reference format. The CD includes Symantec's THINK™ Reference 2.0, working applications with full documentation, product demos for developers and more. See advertisement, this issue: $199. Upgrades $69, e-mail, call or write for info.

BOOKS

The Best of MacTutor, Volume 1: **Sold Out**
The Complete MacTutor, Volume 2: **Sold Out**
The Essential MacTutor, Volume 3: $19.95
The Definitive MacTutor, Volume 4: $24.95
The Best of MacTutor, Volume 5: $34.95
Best of MacTutor Collection, Volumes 3 – 5: $69
Best of MacTutor, Volumes 6, 7, 8 & 9: Not available

DISKS

Source Code Disks: $8 each
Topical Index (1984-1991) on disk: $5

MAGAZINE BACK ISSUES

Volumes 3, 4, 5, 6, 7, 8, 9 and 10: $5 each (subject to availability)

SHIPPING, HANDLING & TAXES

California:
Source disk or single issue: $3
Single book or multiple back issues: $5
Two books: $8 • All other orders: $12

California residents include 8.25% sales tax on all software, disks and books.

Continental US:
Source disk or single issue: $3
Single book or multiple back issues: $7
Two books: $15 • All other orders: $17

Canada, Mexico and Overseas: Please contact us for shipping information.

Allow up to 2 weeks for standard domestic orders, more time for international orders.

PLEASE NOTE

Source code disks and journals from MacTech Magazine are licensed to the purchaser for private use only and are not to be copied for commercial gain. However, the code contained therein may be included, if properly acknowledged, in commercial products at no additional charge. All prices are subject to change without notice.

10% OFF ALL BOOKS! 3RD PARTY PRODUCTS

MACTECH EXCLUSIVES

MacTech Magazine is your exclusive source for these specific products:

NEW! **Ad Lib 2.0** The premier MacApp 3.0 compatible ViewEdit replacement. A powerful user-interface editing tool to build views for MacApp 3.0 and 3.1. Ad Lib allows subclassing of all of MacApp's view classes including adorners, behaviors, and drawing environments. String and text style resources are managed automatically. Alternate display methods, such as a view hierarchy window, allow easy examination of complex view structures. Ad Lib includes source code for MacApp extensions that are supported by the editor – buttons can be activated by keystrokes, behaviors can be attached to the application object, and general purpose behaviors can be configured to perform a number of useful functions. Run mode allows the user to try out the views as they will work in an application. Templates can be created to add additional data fields to view classes. Editing palettes provide fast and easy editing of common objects and attributes. Works with ACI's Object Master (version 2.0 and later) to navigate a project's user interface source code. $195

NEW! **FrameWorks Magazine:** $8/issue, subject to availability.

NEW! **FrameWorks Source Code Disk:** $10/issue, subject to availability.

NEW! **Five Years of Objects CD-ROM:** FrameWorks archives and source code from April 1991 to January 1993, plus selected object-oriented publicly available software and demos. $95

MADACON '93 CD-ROM: The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime. $95

NEW! **MAScript 1.2** adds support for AppleScript to your MacApp 3.0.1 and 3.1 based applications. Make your application scriptable and recordable by building on a tried and tested framework for object model support. MAScript dispatches Apple events to the appropriate objects, creates object specifiers, and makes framework objects like windows and documents scriptable and recordable. Sample application shows you how to begin adding support for scripting and recording. MAScript includes complete source code. Install MAScript by modifying one MacApp source file, then adding another to your project. Future versions of MacApp will incorporate MAScript, so MAScript support you add now will work in the future. $199

NEW! **The Mjølner BETA System** is a software development environment supporting object-oriented programming in the BETA programming language. BETA is uniquely expressive and orthogonal. BETA unifies just about every abstraction mechanism - including class, procedure, function, coroutine, process and exception - into the ultimate abstraction mechanism: the pattern. BETA includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler.

The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. The tools available on Unix: the hyper structure editor supporting syntax directed editing, browsing, etc., and the source code debugger are currently being ported to the Macintosh system. The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later.

Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available) $295

NEW Version! **Savvy 1.1** OSA support includes attachability, recordability, scriptability, coercion, in addition to script execution, idling and i/o. Apple event support includes complex object specifiers, synchronous/asynchronous Apple event handling, and Apple event transactions for clients and servers. The Core Suite of Apple event objects is supported including the application, documents, windows, and files. Documentation includes technology overview, cookbook, and sample code. $250 Savvy now supports MPW 3.1, 3.11 and continues to support 3.01, as well as supporting Metrowerks Code Warrior. **This month only, special offer** - All Savvy versions include free

MAIL ORDER STORE

copy of Savvy QuickTime!

NEW! **More Savvy** includes all Savvy features plus Apple event support for all sub-classes of TEventHandler with extensive view support. Apple event support for text includes text attributes and sub-range specification. Recordability supports additional actions, and coercion includes additional types. Additional client and server Apple events. $450

NEW! **Super Savvy** includes all More Savvy features plus compile, edit, and record scripts using built in script editor. View template editors, like Ad Lib, can attach scripts to view objects and modified scripts are saved with the document. Script action behavior allow quick access for executing and editing scripts attached to views. Text to object specifier coercion plus more. $700

NEW! **Savvy QuickTime** Requires Savvy, More Savvy, or Super Savvy. Includes QuickTime, Apple event and view template support. Movies come out of the box ready to play, edit, and react to Apple events. They can be included in any view structure, including templates, and are displayed in the scrap view. Movie controls include volume, play rate, looping mode, display style, and other characteristics. $250

NEW! **Savvy DataBase** Requires Savvy, More Savvy, or Super Savvy. Available Winter 1994. $250

MacTech Magazine is your exclusive source for available back issues of SFA's magazine, source code disks and assorted CD's. Call for more info and pricing.

BOOKS

Defying Gravity: The Making of Newton Doug Menuez and Markos Kounalakis. An in depth, dramatic account of the story of Newton's creation. It is a technological adventure story; a fascinating case study of the process by which an idea is born and then translated into a product on which careers and fortunes can be made or lost. It is a new kind of business book, one that captures through powerful photojournalism and a fast-paced text, the human drama and risk involved in the invention of a new technology for a new marketplace. 196 pgs., ~~$29.95~~ **$26.95**

The Elements of E-Mail Style by Brent Heslop and David Angell. Learn the rules of the road in the e-mail age. Concise, easy-to-use format explaining essential e-mail guidelines and rules. It covers style, tone, typography, formatting, politics and etiquette. It also outlines basic rules of composition within the special context of writing e-mail and includes samples and templates for writing specific types of e-mail correspondence. 208 pages. ~~$14.95~~ **$13.45**

NEW! **E-Mail Essentials** by Ed Tittel & Margaret Robbins is a hands-on guide to the basics of e-mail, the ubiquitous networks communication system. The book is suitable for both the casual e-mailer and the networking professional, as it covers everything from the installation of e-mail to the maintenance and management of e-mail hubs and message servers. The books explains the fundamental concepts and technologies of electronic mail, featuring chapters on Lotus applications and CompuServe, as well as information on upgrading, automation, message-based applications, and user training. E-mail Essentials is a step-by-step, jargon-free guide that will enable the e-mail user to get the most out of the communication potentials of networking. 250 pp. ~~$24.95~~ **$22.45**

NEW! **Graphics Gems IV** edited by Paul Heckbert Volume IV is the newest collection of carefully crafted, innovative gems. All of the gems are immediately accessible and useful in formulating clean, fast, and elegant programs. The C programming language is used for most of the program listings, although several of the gems have C++ implementations. An IBM or Macintosh disk containing all of the code from all four volumes is included. Includes one 3.5" high-density disk. ~~$49.95~~ **$44.95**

How To Write Macintosh Software by Scott Knaster is a great source for understanding Macintosh programming techniques. Drawing from his years of experience working with programmers, Scott explains the mysteries and myths of Macintosh programming with wit and humor. The third edition, fully revised and updated, covers System 7 and 32-bit developments, and explores such topics as how and where things are stored in memory; what things in memory can be moved around and when they may be moved; how to debug your applications with MacsBug; how to examine your program's code to learn precisely what's going on when it runs. 448 pgs., ~~$28.95~~ **$26.05**

The Instant Internet Guide by Brent Heslop and David Angell. An Internet jump-start – how to access, use and navigate global networks. The Instant Internet Guide equips readers with the tools needed to travel the electronic world. The book highlights the most important sources of Internet news and information and explains how to access information on remote systems. It outlines how to use essential Internet utilities and programs and includes a primer on UNIX for the Internet. 224 pages ~~$14.95~~ **$13.45**

Learn C on the Macintosh by Dave Mark. This self-teaching book/disk package gives you everything you need to begin programming on the Macintosh. Learn to write, edit, compile, and run your first C programs through a series of over 25 projects that build on one another. The book comes with THIN C – a customized version of Symantec's THINK C, the leading programming environment for Macintosh. 464 pages, Book/disk: ~~$34.95~~ **$31.45**

Learn C++ on the Macintosh by Dave Mark. After a brief refresher course in C, Learn C++ introduces the basic syntax of C++ and object programming. Then you'll learn how to write, edit, and compile your first C++ programs through a series of programming projects that build on one another as new concepts are introduced. Key C++ concepts such as derived classes, operator overloading, and iostream functions are all covered in Dave's easy-to-follow approach. Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages, ~~$36.95~~ **$33.26**

Macintosh C Programming Primer Volume I, Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed. This new edition of this Macintosh programming bestseller is updated to include recent changes in Macintosh technology, including System 7, new versions of THINK C and ResEdit, and new Macintosh machines. Readers will learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications. 672 pages, ~~$26.95~~ **$24.25**

Macintosh C Programming Primer Volume II, Mastering the Toolbox Using THINK C by Dave Mark. Volume II picks up where Volume I leaves off, covering more advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs. ~~$26.95~~ **$24.25**

Macintosh Pascal Programming Primer Volume I, Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed. This tutorial shows programmers new to the Macintosh how to use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages ~~$26.95~~ **$24.25**

Macintosh Programming Techniques by Dan Sydow (Series Editor: Tony Meadow). This tutorial and handbook provides a thorough foundation in the special techniques of Macintosh programming for experienced Macintosh programmers as well as those making the transition from DOS, Windows, VAX or UNIX. Emphasizes programming techniques over syntax for better code, regardless of language. Guides the reader through Macintosh memory management, QuickDraw, events and more, using sample program in C++. Disk includes an interactive tutorial, plus reusable C++ code. ~~$34.95~~ **$31.95**

NEW! **Multimedia Authoring Building and Developing Documents** by Scott Fisher addresses the concerns that face anyone trying to create multimedia documents. It offers specific advice on when to use different kinds of information architecture, discusses the human-factors concepts that determine how readers use and retain information, and them applies these findings to multimedia documents, covering the high-level issues concerning planners and authors of multimedia documents as well as those involved in evaluating or purchasing multimedia platforms. Includes one 3.5" high-density disk. ~~$34.95~~ **$31.45**

NEW! **Programming for the Newton Software Development with NewtonScript** by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith. Programming for the Newton: Software Development with NewtonScript is an indispensable tool for Newton programmers. Readers will learn how to develop software for the Newton on the Macintosh from people that developed the course on programming the Newton for Apple Computer. The enclosed 3.5" disk contains a sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK), Apple Computers complete development environment for the Newtons. A Publication of AP Professional May 1994, Paperback, 393 pp. ~~$29.95~~ **$26.95**

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle. This book will introduce you to object-oriented programming, the C++ language, and of course Symantec C++ for the Macintosh. You don't have to be a programmer, or even know anything about programming to benefit from this book. Programming in Symantec C++ for the Macintosh covers everything from the basics to advanced features of Symantec C++. If you are a Think C or Zortech C++ programmer who wants to learn more about object-oriented programming or what's different about Symantec C++, there are chapters specifically for you. Includes helpful examples of C++ code that illustrate object-oriented programs. ~~$29.95~~ **$26.95**

Programming for System 7 by Gary Little and Tim Swihart, is a hands-on guide to creating applications for System 7. It describes the new features and functions of the operating system in detail. Topics covered include file operations, cooperative multitasking, Balloon Help, Apple events, and the File Manager. Numerous working C code examples show programmers how to take advantage of each of these features and use them in developing their

Want more product info? Call us at 310/575-4343.

applications. 384 pages ~~$26.95~~ **$24.25**

ResEdit™ Complete, Second Edition by Peter Alley and Carolyn Strange. With ResEdit, Macintosh programmers can customize every aspect of their interface form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package. ~~$34.95~~ **$31.45**

Sad Macs, Bombs, Disasters and What to Do About Them by Ted Landau comes to the rescue with your Macintosh problems. From fractious fonts to the ominous Sad Macintosh icon, this emergency handbook covers the whole range of Macintosh problems: symptoms, causes, and what you can do to solve them. 640 Pages ~~$24.95~~ **$22.45**

Software By Design: Creating User Friendly Software by Penny Bauersfeld (Series Editor: Tony Meadow). This excellent reference provides readers with a thorough how-to for designing software that is easy to learn, comfortable to operate and that inspires user confidence. Written from the perspective of Macintosh, but compatible with all platforms. Stresses user input from initial design, through prototyping, testing and revision. Provides tools for analyzing user needs and test responses. Includes exercises for sharpening user-oriented design skills. ~~$29.95~~ **$26.95**

NEW! **Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++** is the Taligent approach to object-oriented design. The Taligent Operating Environment is the first commercial software system based entirely on object-oriented technology. Taligent's Guide to Designing Programs is a developer's-eye view of this system. It introduces new concepts of programming and empowers developers to create software more productively. Out of their direct experience in developing the system, the authors focus on global issues of object-oriented design and writing C++ programs, and the specific issues of programming in the Taligent Operating Environment. Taligent's Guide to Designing Programs assumes the reader is an experienced C++ programmer, and proceeds from there to fully explore "the Taligent way' of programming. ~~$19.50~~ **$17.55**

Writing Localizable Software for the Macintosh by Daniel R. Carter. 469 pages. ~~$26.95~~ **$24.25**

THE APPLE LIBRARY

HyperCard Stack Design Guidelines by Apple Computer, Inc. is an essential book for everyone who creates Apple HyperCard stacks, from beginners to commercial developers. It covers the basic principles of design that, when incorporated, make HyperCard stacks effective and usable. Topics include guidelines, navigation, graphic design and screen illustration, text in stacks, music and sound, a sample stack development scenario, collaborative development, and the Stack Design Checklist. 240 pages, ~~$21.95~~ **$19.95**

Inside AppleTalk by Gursharan S. Sidhu, Richard F. Andrews and Alan B. Oppenheimer. Apple Computer, Inc. 650 pages, ~~$34.95~~ **$31.45**

Inside Macintosh: AOCE Application Interfaces by Apple Computer, Inc. shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers. Nearly every Macintosh application program can benefit from the addition of some of these features. This book shows how you can add electronic mail capabilities to your application, write a messaging application or agent, store information in and retrieve information from PowerShare and other AOCE catalogs, add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOCE catalogs, add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection. ~~$40.45~~ **$36.40**

Inside Macintosh: AOCE Service Access Modules by Apple Computer, Inc. describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service. This book shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOCE templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages. ~~$26.95~~ **$24.25**

NEW! **Inside Macintosh: CD-ROM** by Apple Computer, Inc. Inside Macintosh® is the essential reference for programmers, designers, and engineers for creating applications for the Macintosh family of computers. Inside Macintosh CD-ROM collects more than 25 volumes in electronic form, including: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Now programmers will be able to access over 16,000 pages of the information they need directly from their computers. Hypertext linking and extensive cross referencing across volumes allows programmers to search and explore this library in ways that are unique to the electronic medium. Every Macintosh programmer will regard Inside Macintosh CD-ROM as their most important resource. $99.95

Inside Macintosh: Devices by Apple Computer, Inc. describes how to write software that interacts with built-in and peripheral hardware devices. With this book,. you'll learn how to write and install your own device drivers, desk accessories, and Chooser extensions; communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager; communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers; and communicate with serial devices using the Serial Driver. ~~$29.95~~ **$26.95**

Inside Macintosh: Files by Apple Computer, Inc. describes the parts of the operating system that allow you to manage files. It shows how your application can handle the commands typically found in a File menu. It also provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs, ~~$29.95~~ **$26.95**

Inside Macintosh: Interapplication Communication by Apple Computer, Inc. shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases. ~~$34.95~~ **$31.45**

Inside Macintosh: Imaging by Apple Computer, Inc. covers QuickDraw and Color QuickDraw. The book includes general discussions of drawing and working with color. It describes the structures that hold images and image information, and the routines that manipulate them. It also covers the Palette, Color, and Printing Managers, and the Color Picker, Color Matching, and Picture

Want more product info? E-mail us at productinfo@xplain.com

Utilities. ~~$26.95~~ **$24.25**

Inside Macintosh: Macintosh Toolbox Essentials by Apple Computer, Inc. covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel". This book describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages ~~$34.95~~ **$31.45**

Inside Macintosh: More Macintosh Toolbox by Apple Computer, Inc. covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels are covered in this volume. The managers discussed include Help, List, Resource, Scrap and Sound. ~~$34.65~~ **$31.45**

Inside Macintosh: Memory by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to manage memory. It provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages, ~~$24.95~~ **$22.45**

Inside Macintosh: Networking by Apple Computer, Inc. describes how to write software that uses AppleTalk networking protocols. It describes the components and organization of AppleTalk and how to select an AppleTalk protocol. It provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others. ~~$29.95~~ **$26.95**

Inside Macintosh: Operating System Utilities by Apple Computer, Inc. describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system. Everyone who programs the Macintosh should read this book! It will show you in detail how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors. ~~$26.05~~ **$23.45**

Inside Macintosh: Overview by Apple Computer, Inc. is the first book that people who are unfamiliar with Macintosh programming should read. It gives an overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library. Inside Macintosh: Overview also covers various programming tools and languages, compatibility guidelines and an overview of considerations for worldwide development. 176 pages, ~~$22.95~~ **$20.65**

Inside Macintosh: PowerPC Numerics by Apple Computer, Inc. describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers. The numerics environment conforms to the IEEE standard 754 for binary floating-point arithmetic. This book provides a description of that standard and shows how RISC Numerics compiles with it. This book also shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language. ~~$28.95~~ **$26.00**

Inside Macintosh: PowerPC System Software by Apple Computer, Inc. describes the new process execution environment and system software services provided with the first version of the system software for Macintosh on PowerPC computers. It

MAIL ORDER STORE

contains information you need to know to write applications and other software that can run on the PowerPC. PowerPC System Software shows in detail how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager. ~~$24.95~~ **$22.45**

Inside Macintosh: Processes by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks. It shows in detail how you can use the Process Manager to get information about processes loaded in memory. It is also a reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages, ~~$22.95~~ **$20.65**

Inside Macintosh: QuickTime by Apple Computer, Inc. is for anyone who wants to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications. This book describes all of the QuickTime Toolbox utilities. In addition, it provides the information you need to compress and decompress images and image sequences. ~~$29.95~~ **$26.95**

Inside Macintosh: QuickTime Components by Apple Computer, Inc.covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers. ~~$34.95~~ **$31.45**

Inside Macintosh: Sound by Apple Computer, Inc. describes the parts of the Macintosh system software that allow you to manage sounds. It contains information that you need to know to write applications and other software that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations. ~~$26.95~~ **$24.25**

Inside Macintosh: Text by Apple Computer, Inc. describes how to perform text handling, from simple character display to multi-language processing. The Font, Script, Text Services, and Dictionary Managers are all covered, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources. ~~$39.95~~ **$35.95**

Inside Macintosh: QuickDraw™ GX Library by Apple Computer, Inc. is the powerful new graphics architecture for the Macintosh. Far more than just a revision of QuickDraw, QuickDraw GX is a unified approach to graphics and typography that gives programmers unprecedented flexibility and power in drawing and printing all kinds of shapes, images, and text. This long-awaited extension to Macintosh system software is documented in a library of books that are themselves an extension to the new Inside Macintosh series. The QuickDraw GX Library is clear, concise, and organized by topic. The books contain detailed explanations and abundant programming examples. With extensive cross-references, illustrations, and C-language sample code, the QuickDraw GX Library gives programmers fast and complete reference information for creating powerful graphics and publishing applications with sophisticated printing capabilities. The first two volumes in the QuickDraw GX Library are:

Inside Macintosh: QuickDraw GX Objects by Apple Computer, Inc. introduces QuickDraw GX and its object structure, and shows programmers how to manipulate objects in all types of programs. ~~$26.95~~ **$24.25**

Inside Macintosh: QuickDraw GX Graphics by Apple Computer, Inc. shows readers how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities. It also demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes. ~~$26.95~~ **$24.25**

NEW! **Inside Macintosh®: QuickDraw™ GX Environment and Utilities** A companion to QuickDraw™ GX Objects, this book contains programming information useful to any developer writing QuickDraw GX applications. It describes QuickDraw GX memory management, error handling, debugging, and mathematical functions, as well as conversion from QuickDraw to QuickDraw GX. ~~$29.95~~ **$26.95**

NEW! **Inside Macintosh®: QuckDraw™ GX Printing** This book is essential for any developer whose QuickDraw™ GX application supports printing. It shows how to support the new printing features of QuickDraw GX, including desktop printers and expandable printing dialog boxes. QuickDraw GX Printing also shows how to use printing-related objects to add custom panels to printing dialog boxes and to create custom page formats. ~~$26.95~~ **$24.25**

NEW! **Inside Macintosh®: QuckDraw™ GX Printing Extensions and Drivers** Any developer who wants to create extensions to the application printing capabilities of QuickDraw™ GX, or who needs to write a printing device driver that works with QuickDraw GX needs this book. QuickDraw GX Printing Extensions and Drivers describes how to create printing extensions and printer drivers, and provides a complete reference to the messages, functions, and resources that they use. ~~$29.95~~ **$26.95**

NEW! **Inside Macintosh®: QuckDraw™ GX Programmer's Overview** This book provides an introduction to QuickDraw™ GX, providing an overview of the QuickDraw GX environment from a developer's perspective. It introduces the QuickDraw™ GX programming and runtime environments, the relationship between QuickDraw GX and the rest of the Macintosh® systems software and the relationship between QuickDraw GX and Macintosh applications. The key elements of QuickDraw GX programming, data structures, object types, and functions used most frequently by QuickDraw GX developers are also covered. After a general introduction, this book provides readers with a series of practical examples demonstrating how to approach programming with QuickDraw GX. ~~$24.95~~ **$22.45**

NEW! **Inside Macintosh®: QuckDraw™ GX Typography** This book is essential for any developer who uses QuickDraw™ GX to manipulate text. It shows how to use QuickDraw GX objects to handle all kinds of text – from plain, unstyled text to complex, mixed-direction and multi-language text with sophisticated stylistic and typographic variations. QuickDraw GX Typography shows how to create and manipulate the three different types of text shapes supported by QuickDraw GX including text shapes, glyph shapes, and layout shapes. ~~$29.95~~ **$26.95**

NEW! **Inside Macintosh:** X-Ref. by Apple Computer, Inc. is an index for Inside Mac. ~~$12.95~~ **$11.65**

LANGUAGES

CodeWarrior™ CD by Metrowerks comes in two versions – Bronze and Gold. These CDs contain the CodeWarrior development environment including C++, C and Pascal compilers; high-speed linkers; native-mode interactive debuggers; and a powerful new application framework called PowerPlant for rapid Macintosh development in C++. Bronze generates 680x0 code. Gold generates both 680x0 and PowerPC code. All versions are a 3 CD subscription over a 1-year period. Bronze: $99, Gold: $399. **Bronze comes with a 6-month MacTech subscription. Gold comes with a 1-year subscription. Both at no additional charge!**

NEW! **Geekware** by Metrowerks is here! In high school, they called you a computer geek. Now, they work at burger joints and wear polyester uniforms. And you don't. Wear it to your favorite burger joint. $24.95

FORTRAN by Language Systems is a full-featured ANSI standard FORTRAN 77 compiler that runs in the Macintosh Programmers Workshop (MPW). All major VAX extensions are supported as well as all major features of Cray and Data General FORTRAN. FORTRAN creates System 7 savvy applications quickly and easily. Compiler options specify code generation and optimization for all Macintoshes, including special optimizations for 68040 machines. Error messages are written in plain English and are automatically linked to the source file. The runtime user interface of compiled FORTRAN programs is fully customizable by programmers with any level of Macintosh experience. $595. w/o MPW: $495. Corporate 5 pack $1575

FORTRAN 77 SDK for Power Macintosh by Absoft includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment. The compiler is a full ANSI/ISO FORTRAN 77 implementation and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements. MRWE, Absoft's application framework libraries, is included as is the MIG graphics library for quick creation of plots and graphs. The native Macintosh PPC toolbox is fully supported. Absoft's Fx debugger can debug intermixed FORTRAN 77,C, C++, PPC assembler. The compiler, linker, and debugger all run as native PPC tools and produce native Macintosh PPC executables. $699

MacFortran® II V3.3 is a VAX/VMS compatible, full ANSI/ISO FORTRAN 77 compiler including all MIL-STD 1753 extensions. Acknowledged to be the fastest FORTRAN available for Macintosh, MacFortran II is bundled with the latest version of Macintosh Programmer's Workshop (MPW), and includes SourceBug (Apple's source level symbolic debugger) and SoftwareFPU (a math co-processor emulator). Also included is Absoft's Macintosh Runtime Window

Want more product info? Call us at 310/575-4343.

Environment (MRWE) application framework (with fully documented source code as examples) and MIG graphics library. MacFortran II v3.3 features improved 68040CPU support and is fully compatible with Power Macintosh under emulation. Documentation includes special sections devoted to use of MacFortran II with the MPW editor and linker, implementation of System 7 features, and porting code to the Macintosh from from various mainframes and Unix workstation platforms. $595

REM GOSUB LET *NS* BASIC

BASIC for the Newton is BASIC for the Newton! From NS BASIC Corporation, it is a fully interactive implementation of the BASIC programming language. It runs entirely on the Newton - no host is required. It includes a full set of functions and data types, hand-written input, windows, buttons and extensions to take advantage of the Newton environment. Applications can create files or access the built-in soups. Applications can also access the serial port for input and output. Work directly on the Newton, or through a connected Mac/PC and keyboard. NS BASIC includes a 150 page pocket sized manual. $99

Q·K·S

SmalltalkAgents™, a superset of the Smalltalk language, is fully integrated with Macintosh, incorporating design features specifically for the RISC and Macintosh System 7 architecture. SmalltalkAgents is a true object oriented workbench that includes an incremental and extensible compiler, an array of design and cross reference tools, preemptive interrupt driven threads and events, an extensive class library including classes for general programming, classes for the Macintosh user interface and classes for the Macintosh operating system. Integration of components in enterprise systems is simplified with the network, telecommunication, and inter-application communication libraries. The SmalltalkAgents' extensive class library and add-on components make it especially well suited as a development workbench for custom applications in business, education, science, engineering, and academic research. $695

SYMANTEC.™

Symantec C++ for Macintosh is an object oriented development environment designed for professional Macintosh programmers. Symantec C++ features powerful object-oriented development tools within a completely integrated environment. The C++ compiler, incremental linker, THINK Class Library, integrated browser, and automatic project management give Symantec C++ fast turnaround times. This product supports multiple editors and translators, so you can use your favorite tools and resource editors as well as scripts you've written within the environment. And with ToolServer, you'll be able to customize menus and attach scripts based on Apple events, AppleScript, and MPW Tools. The built-in SourceServer provides a source code control system, allowing teams of programmers to solve tough problems faster. With SourceServer, you'll always know you're working on the latest version. And you'll have old versions at your fingertips when code "breaks" and you need to look back at modifications. Product Contents: Three high density disks, an 832-page user manual, a 568-page THINK Class Library and a 100-page C++ Compiler Guide. $369

THINK C by Symantec Corporation. THINK C is easy to use and highly visual, making it the No. 1 selling Macintosh programming environment. Enhancements make this product faster and more versatile than ever, improving your productivity with more powerful project management, a full set of tools, and script support for major script-based languages. With the THINK environment, you spend less time on routine programming tasks due to an extremely fast compiler and incremental linker. In addition, the automatic project manager saves you time by tracking changes to your files and recompiling only those that have changes. All the tools you need – a multi-window editor, compiler, linker, debugger, browser, and resource editor – are completely integrated for speed and ease of use. One of the most valuable of these tools is the THINK Class Library, a set of program building blocks that gives you a head start in writing object-oriented applications. And with the new open architecture, you can use your favorite tools, resource editors, and scripts within the environment. THINK C is the logical next step for programmers who have worked in HyperCard or other script-based development environments. The environment supports AppleScript, Apple events, and Frontier, so you can link and automate complex, multi-project operations. Product Contents: Four Macintosh disks, an 832-page user manual, and a 568-page THINK Class Library Guide. $219

THINK Pascal v. 4.0 by Symantec Corporation. Professionals and students will welcome this version of THINK Pascal. It is fully integrated for rapid turnaround time and lets you take advantage of System 7 capabilities. Features include support for large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking. Product Contents: Four Macintosh disks, a 562-page user manual, and a 498-page object-oriented programming manual. $169

UTILITIES

NEW! **BBEdit 3.1** from Bare Bones Software is now better than ever. In addition to being Accelerated for Power Macintosh, this powerful, intuitive text editor offers integrated support for THINK C 7.0, Metrowerks CodeWarrior, THINK Reference 2.0 and MPW ToolServer. Version 3.1 adds even more capability, including "soft" wrapping of text on screen and numerous refinements and improvements to the user interface. BBEdit's many features include: Integrated PopupFuncs™ technology for speedy navigation of source code files (C, C++, Pascal, Rez, 68K Assembler, and Fortran), unique 'Find Differences' command (BBEdit can find differences between projects and folders as well as files), support for Macintosh Drag and Drop for editing and other common tasks, PowerTalk support for reading, sending and composition of PowerTalk mail, scripting via any OSA compatible scripting language including AppleScript and Frontier 3.0, and fast search and replace with optional "grep" matching and multi-file searching. BBEdit's robust feature set and proven performance and reliability make it the editor of choice for professionals and hobbyists alike. $119

NEW! **Call Tree** by Barking Dog Software Co is a quick way to understand the structure of complex programs. Utilizing Drag & Drop, the user can produce a call tree analysis of C source code files to assist in optimizing, structuring and a general understanding of software systems. Call Tree is stand-alone and works with CodeWarrior, Think C, Symantec, MPW and most other C source text files. Call Tree is simple to use and can save many hours of tedious work. $149.

CLImate by Orchard Software is a command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders. It can start applications, format disks, restart your computer, etc. CLImate supplements the Finder. It includes a BASIC interpreter that lets you script your Macintosh without AppleScript. The interpreter includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc... CLImate implements wildcard characters, enabling you to work on groups of files. Use CLImate instead of MPW to manage your projects. CLImate is an application occupying 70K disk space. It comes bundled with sample programs and full documentation. $59.95

CMaster 2.0 by Jersey Scientific installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh, and enhances the editor. Use its function popup to select a function and CMaster takes you right to it. Other features include multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts. Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too! $129.95

Cron Manager by Orchard Software implements the UNIX Cron facility. It can open any Macintosh file on a given date and time. By creating an alias, renaming it to the date and time to open, and moving it into the special Cron Events Folder, Cron Manager will open it. Cron Manager is a control panel that creates the special Cron Events Folder inside your System Folder. It is completely transparent to the user. It works like the Startup Items folder, only smarter. It works with any Macintosh file: if you can double-click to start it, Cron Manager can open it. $26.95. Cron Manager bundled with CLImate, $59.95

dtF is a true relational database system for Apple Macintosh computers. dtF provides a powerful choice for developers who want to create database centered applications with no performance trade-offs. dtF features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX. The C/C++ API is identical and fully portable cross all supported platforms. Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free. Tools are included for importing, exporting, creating and managing databases and users. Supported development environments include: Symantec, MPW, MetroWerks and more. Mac/SDK: $695

InstallerPack™ by StepUp Software is a package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts. Compression formats supported are Compact Pro & Diamond. Each atom also available separately: $219

Last Resort Programmer's Edition records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk. This is especially useful for program testing & debugging, and for technical support and help desks. If something goes wrong (because of a power failure, system crash, forgetting to save or deleting lines) and you lose a word, phrase, or document you can look in the Last Resort keystroke file and recover what you typed. Last Resort is also useful for technical support personnel, when they have to ask "What was the last thing you did before...?" $74.95

NEW! **LJ Profiler** by Lars Jordebo Datakonsult supports profiling of C++ 68K and Power PC applications compiled with CodeWarrior, CFront or

Want more product info? E-mail us at productinfo@xplain.com

MAIL ORDER STORE

Symantec C++. Based on active profiling, i.e. profiling code called at function enter and exit, the browser application lets you follow call chain timings in hierarchical views or separate windows. Collect, organize, compare and save profiling data from different versions of your application into a project. Scriptable and recordable with full access to most internal data structures. Optional remote profiling and tracking of segment and stack usage. Full source code to what you link into your application. $295.

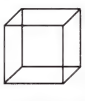
LS Object Pascal CD includes the world's first Object Pascal compiler for Power Macintosh. 100% compatible with Apple's MPW Pascal, LS Object Pascal combines the best of Apple's native development tools with innovative new technology developed at Language Systems. Compiler options specify 68K or native PowerPC code generation. Included on the CD are: LS Object Pascal compiler, Universal Pascal Toolbox interfaces, fully loaded MPW 3.3.1, 68K and PowerPC source debuggers, PowerPC assembler, online documentation, Macintosh Tech Notes, and a special version of AppMaker by Bowers Development that generates native Pascal source code. The beta release includes upgrades to v1.0 when it becomes available. $399

Spellswell 7 1.0.4 is an award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness). Spellswell 7 checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double double word errors, abbreviation errors, mixed case errors, extra spaces between words, a/an before vowel/consonant, etc... MacTech orders include developer kit with Writeswell Jr., a sample Apple Events Word Services word-processor and its source code. $74.95

MacAnalyst/Expert by Excel Software supports software engineering methods with the capabilities of MacAnalyst plus state transition diagrams, state transition tables, decision tables and process activation tables. An integrated requirement database provides traceability from requirement statements to analysis or design diagrams, code or test procedures. This tool is well suited to the analysis and design of real-time or requirements driven projects. Demo $79, Product $1595 **MacAnalyst** Demo $79, Product $995

MacDesigner/Expert by Excel Software supports software engineering methods with the capabilities of MacDesigner plus multi-task design. An integrated requirement database provides traceability from requirement statements to design diagrams, code or test procedures. This tool is well suited to design or maintenance of real-time, multi-tasking software projects. Demo $79, Product $1595. **MacDesigner** Demo $79, Product $995

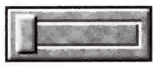
MacA&D by Excel Software combines the capabilities of MacAnalyst/Expert and MacDesigner/Expert into a single application. It supports structured analysis and design, object-oriented analysis and design, real-time extensions, task design, data modeling, screen prototyping, code editing and browsing, reengineering, requirement traceability, and a global data dictionary. Demo $149, Product $2995



MacWireFrame by Amplified Intelligence. Create your own virtual reality application with MacWireFrame, a virtual reality application frame work. Includes a complete library of object oriented graphics routines, its own easy to understand application frame work (similar to MacApp or TCL but a lot easier to understand), plus an example application program that lets you start solid modeling right away. Comes complete with fully documented source code. All new purchases will be guaranteed a $49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0. Due to the overwhelming response the special price offer has been extended for a little while longer. **Special Offer:** ~~$299.00~~ **$75!!!!**

The Memory Mine™ by Adianta is a stand alone debugging tool for Macintosh and native PowerPC. Programmers can monitor heaps, identify problems such as memory leaks, and stress test applications. Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more... Allocate, Purge, Compact, and Zap memory let users stress test all or part of a program. Source code is not needed to view heaps. It works on Macintoshes with 68020 or later and System 7.0 or later. $99

NEW! **PictureCDEF 1.3** by Paradigm Software is a professional-level CDEF for creating custom graphical buttons (8-64 pixels). PictureCDEF is used in products by Adobe, ProVue, STF Technologies and others. It is multi-monitor and bit-depth sensitive. The button graphic (cicn, ResEdit) can be changed at runtime and even animated with a call-back routine. Create distinct buttons in seven variations: MultiState, PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton. Position the optional button title at left, bottom or right, or follow the system text direction for



international support. Manual, sample code and MacApp 3.0 support included. Full source code: $95.00 Object code: $45.00.

Qd3d/3dPane/SmartPane source code bundle by Vivistar Consulting. **Qd3d 2.0:** Full featured 3d graphics. Points; lines; polygons; polyhedra; Gouraud shading; z-buffering; culling; depth cueing; parallel, perspective, and stereoscopic projections; performance enhancing "OnlyQD" and "Wireframe" modes; full clipping; pipeline access; animation and model interaction support; and a "triad mouse" to map 2d mouse movement to 3d. **3dPane 2.0:** Integrates Qd3d with the TCL and provides a view orientation controller. **SmartPane:** Offscreen image buffering, flicker free animation, and QuickTime movie recording. For use with Qd3d/3dPane or in 2d settings. All work with C++ compilers or ThinkC 6. $192

NEW! **QC™** by Onyx Technology, is a system extension that stress tests code during runtime for common and not-so-common errors. Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking. QC is extremely user friendly for the non-technical tester yet offers an API for programmers who want precise control over testing. $99

NEW! **QUED/M 2.7** by Nisus Software, is a programmer's text editor which has defined the industry standard for speed and efficiency. With integrated support for Symantec C/C++, Metrowerks CodeWarrior, and MPW, QUED/M offers unrivaled usefulness for the Macintosh developer. In addition to supporting all the major development environments on the Macintosh, QUED/M offers dozens of powerful editing features, including unlimited undo and redo, UNIX style GREP searching, macro language, scripting, text folding, text sorting, file comparison and merging, Toolbox lookup, ten editable/appendable clipboards, line numbering, markers, displaying text as ASCII codes, vertical and horizontal screen splitting, plus much more. QC is Also available in Japanese. **$149**

ScriptGen Pro™ by StepUp Software is an Installer script generator which requires no programming or knowledge of Rez. Supports StepUp's InstallerPack, StuffIt compression, custom packages, splash screens, network installs, Rez code output, importing resources, and AppleEvent link w/MPW: $169

SoftPolish by Language Systems is a development tool that helps software developers avoid embarrassing spelling errors, detect incorrect or incompatible resources and improve the appearance of their Macintosh software. SoftPolish examines application resources and reports potential problems to a scrolling log. Independent of any programming language or environment, SoftPolish improves the quality of any Macintosh program. $169

NEW! **Spyer** by InCider is a simple operated tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed. The recorded data can be saved in files for future use. Spyer works as a background process with any Macintosh application and is triggered by user defined Hot Keys. Spyer enables the "Continuous Redo" utility and is especially useful for software testing and demonstration. $39

StoneTable: A library replacing all functions found in list manager plus: variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior C. (all prices per developer) $150 first compiler, additional compilers $50

Stone Table Extra: Additional functions for StoneTable. Drag selected cells within table or to other tables; optionally add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells. Requires StoneTable. (all prices per developer) $50 first compiler, additional compilers $25

NEW! **StoneTable and StoneTableExtra for PowerPC:** Same functionality as 68K libraries. Versions for MPW C and CodeWarrior C. Must have 68K libraries. (all prices per developer) StoneTable $100, StoneTableExtra $25

NEW! **Version Control** by Barking Dog Software Co. provides easy to use source code control for CodeWarrior, Think C, Symantec and other text files. Drag & Drop your checkins for speed. Recover any individual version or an entire release using the snapshot feature. Tracks the who, when and why of source changes and makes backing out changes a simple task. Version Control is stand-alone, requiring no other tools to operate. $199 for a single user. Multiuser licenses are; 2 users $249, 3-5 users $349, and 6-10 users $499.

Want more product info? Call us at 310/575-4343.

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

Absoft	51
Acuson	83
Adianta Inc.	32
Aladdin Knowledge Systems Ltd.	5
APDA, Apple Computer, Inc.	39
Apple Developer University	41
BareBones Software	87
Barking Dog Software Co.	73
Celestin Company	77
Creative Solutions	21
DataPak Software, Inc.	72
Dell	13
Digitool, Inc.	18
dtF Americas, Inc.	23
Foundation Solutions	65
Full Moon Software	74
Graphic Magic Inc.	78
Graphical Business Interfaces Inc.	55, 30 & 31
Heizer Software	48
Jasik Designs	29
Language Systems	63
The Mac Group	57
MacTech CD-ROM, Vol. 1-9	88
MacTech Mail Order Store	61
MacXperts	82
Mainstay	1
Manpower Technical Services	82
Mathemaesthetics, Inc.	6
Metrowerks	IFC
Microsoft Corporation	15
MindVision Software	17
Neologic Systems	25
Nisus	62
Onyx Technology	75
OPtions Computer Consulting	41
PACE Anti-Piracy	45
Parallel Software	82
Prograph International	11
Quasar Knowledge Systems	IBC
Rainbow Technologies	9
Random House Electronics	22
Ray Sauers Associates	66
Richey Software Training	76
Scientific Placement	82
Sierra Software Innovations	BC
SNA, Inc.	68, 69
Software Designs Unlimited	19
Summit Software Company	47
The Soft Publishing Group	32
TSE International	28
Water's Edge Software	71
Apple's World Wide Developer Conference	37

LIST OF PRODUCTS

Apple's World Wide Developer Conference • Apple's World Wide Developer Conference	37
Apprentice • Celestin Company	77
BasicScript • Summit Software Company	47
BBEdit 3.1 • BareBones Software	87
Books • Random House Electronics	22
BroadCast™ • SNA, Inc.	68
C++ For Power Macintosh • Absoft	51
Call-Tree • Barking Dog Software Co.	73
Cataloger™ • Graphical Business Interfaces Inc.	30 & 31
CodeWarrior™ • Metrowerks	IFC
CXBase Pro • TSE International	28
The Debugger V2 • Jasik Designs	29
Debugging Services • The Mac Group	57
The Dell® Developer System • Dell	13
Developer VISE 3.0 • MindVision Software	17
Directory of Suppliers to the Software Publishing Industry • The Soft Publishing Group	32
dtF, The Relational Database System • dtF Americas, Inc.	23
DragInstall 1 • Ray Sauers Associates	66
EHelp 4.0 • Foundation Solutions	65
FACESPAN • Software Designs Unlimited	19
4D TOOLKIT 2.0 • OPtions Computer Consulting	41
IcePick 3.0™ • Sierra Software Innovations	BC
Inside Out II® • Sierra Software Innovations	BC
LS Object Pascal/PPC • Language Systems	63
MacApp Programmers • MacXperts	82
MacEncrypt™ • PACE Anti-Piracy	45
MacForth Plus 4.2 • Creative Solutions	21
MacHASP • Aladdin Knowledge Systems Ltd.	5
Macintosh Common LISP • Digitool, Inc.	18
MacNosy • Jasik Designs	29
Mac OS SDK • APDA, Apple Computer, Inc.	39
MacRegistry™ • Scientific Placement	82
neoAccess™ • Neologic Systems	25
OP2CPlus • Graphic Magic	78
p2cII™ • Sierra Software Innovations	BC
PAIGE™ • DataPak Software, Inc.	72
PatchWorks™ • SNA, Inc.	69
Power MacForth • Creative Solutions	21
Power PC Training • Apple Developer University	41
Prograph CPX • Prograph International	11
Programmer Training • Richey Software Training	76
QC: The Macintosh Testing Solution • Onyx Technology	75
QUED/M 2.7 • Nisus	62
Recruitment • Acuson	83
Recruitment • MacXperts	82
Recruitment • Manpower Technical Services	82
Recruitment • Parallel Software	82
Recruitment • Scientific Placement	82
Resorcerer® 1.2 • Mathemaesthetics, Inc.	6
SALESBASE 2.1™ • Sierra Software Innovations	BC
ScriptWizard • Full Moon Software	74
Sentinel® • Rainbow Technologies	9
Sierra Consulting Group • Sierra Software Innovations	BC
SmalltalkAgents® • Quasar Knowledge Systems	IBC
Solution Tools • Heizer Software	48
TableIt!™ • Graphical Business Interfaces Inc.	55
Template Constructor™ • Graphical Business Interfaces Inc.	30 & 31
The Memory Mine™ • Adianta Inc.	32
Tool Plus™ 2.5 • Water's Edge Software	71
VIP-C • Mainstay	1
Windows™ 95 • Microsoft Corporation	15

TIPS & TIDBITS

By Scott T Boyd, Editor

TIP OF THE MONTH

Doing Passwords the Simple Way

A common problem for people writing multi-user software for the Mac is the need to have a username/password dialog which allows the user to type the password without actually printing the real characters. Apple's inimitable Tim Dierks released a snippet appropriately entitled "Password" which shows how to accomplish this in a variety of ways, most of which involve keeping one buffer which holds what the user actually types and another which contains the 'display' text (usually a line of bullets).

The following snippet shows how to accomplish this by replacing the QuickDraw bottlenecks. The beauty of this approach is that there is no messy buffer-update code needed for handling Cut/Copy/Paste, and text selection with the mouse is handled correctly. You may, however, want to prevent people from cutting or copying text to the clipboard, since if that text is pasted into any other TextEdit area it will display the password in normal text.

```
#ifdef __cplusplus
extern "C" {
#endif
```

PasswordDrawProc

```
        void    PasswordDrawProc (void)
        {
                Ptr textBuff;
                short byteOff;
                short byteCnt, i;

                // load the correct parameters
                asm     {       move.l  A0, textBuff
                                move.w  D0, byteOff
                                move.w  D1, byteCnt
                        }
                for (i = 1; i <= byteCnt; i++)
                  DrawChar ('•');
        }
```

PasswordMeasureProc

```
        void    PasswordMeasureProc (void)
        {
                short   textLen, offset;
                Ptr     textBuffer;
                short   width;

                // load the correct parameters
                asm     {       move.w  D0, textLen
                                move.w  D1, offset
                                move.l  A0, textBuffer
                        }
                width = CharWidth('•') * textLen;

                // return the width of the text
                asm     {       move.w  width, D1       }
        }

        TEHandle MakePasswordTextBox (void)
        {
                Rect     viewRect, destRect;
```

Continued on page 87

Send us your tips and we'll wash your car, or we might just pay you $25 for each tip we use, or $50 for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to all of us here at editorial@xplain.com. See page two for our other addresses.

So Easy To Forget These Things

I asked the question recently, "When debugging using Think C, after hitting the programmer's switch and landing in MacsBug, how do I get back up to Think C's source level debugger?"

The answer, while less than what I was looking for, was nonetheless useful. Think C's debugger polls for Option-Command-Shift-Period. Use that instead of the programmer's switch and you will be able to break into the debugger as long as the program is stuck in your own code, and not in the operating system somewhere.

– Jeffrey A. Lomicka, Marlboro MA, lomickaj@prnsys.enet.dec.com

What'll They Call The Next One? Post-Modern Memory Manager?

So you're trying to figure out whether a zone is a Modern Memory Manager zone? Why? Oh, never mind, you've certainly got a good reason (you do, don't you?), so here's how:

```
#define IsNewStyleZone(z)           \
            ((z)->heapType & 0x02)

if (IsNewStyleZone(LMGetSysZone()) {
   ...whatever...
```

– Dave Falkenburg, Apple

SmalltalkAgents®

A Dynamic Object-Oriented Development Environment

Call now to receive our customer's own descriptions of why they're choosing AO/S to deliver their products!

Agents Object System (AO/S): Delivering Component-based Technology

SmalltalkAgents is a set of development and authoring components built on the Agents Object System™ (AO/S™). The AO/S Component Toolbox™ is a portable layer of abstraction between AO/S Components and host system services. The AO/S delivers a user extensible home/container that transparently wrappers a variety of component technologies including OpenDoc, OLE, and OSA Scripting.

AO/S: Core Technologies

The AO/S Core Technologies include rich intelligent component and agent support, vendor independent database services, concurrent application support (much like the Apple Finder), separate user interface threads enabling both tethered development and robust "memory protected" application deployment, and a shareable object system, all of which makes it ideal for high-performance client and server applications.

Scalability

Unlike other" application builders", the AO/S Product Family (including SmalltalkAgents Professional and VisualAgents) allows you to go from user-level scripting to full professional-level system development and back. SmalltalkAgents is scalable, from the creation of small and simple applications to sophisticated and complex multi-user systems.

Based on industry-proven Smalltalk language

Smalltalk is the pure Object-Oriented (OO) language which defined the OOP concept, and is now the corporate language of choice for new business applications and sophisticated client/server systems.

What this means to you!

SmalltalkAgents Professional, built on the unique AO/S architecture, is a modern, innovative new development system that blends the best of component-based architecture with the best in object-oriented development technologies.

What the press and customers say:

"...... the most unique and innovative Smalltalk development environment..."
Object Magazine, October 1994

"I must say, this is NOT the Smalltalk of several years ago, and I AM TRULY IMPRESSED WITH WHAT YOU HAVE ACCOMPLISHED!"
E.S. Taylor, Independent Consultant, New York, NY

"SmalltalkAgents has been my favorite Smalltalk environment and I believe it represents the best chance Smalltalk has of becoming an accepted grass roots language for developers."
David Scott, General Atomics, San Diego, CA

Mac OS

AO/S™

Agents Object System
MACINTOSH

Don't be passed by, catch the Component Technology wave with SmalltalkAgents! To gain the Power Macintosh advantage now, call:

1-800-296-1339
or 301-530-4853 (info@qks.com)

QKS

Buy now to receive free upgrade to SmalltalkAgents Pro v2.0!

Quasar Knowledge Systems • 9818 Parkwood Drive • Bethesda, Maryland 20814-4028 USA • Phone: 301-530-4853 • Fax: 301-530-5712

Sierra Software Innovations

Six Years of Quality Products and Services

SALESBASE 2.1™
Fully Integrated Sales Automation Solution for Large Corporations

Inside Out II®
Industrial Strength Multi-User Relational Database Engine

p2cII™
Advanced Object Pascal to C++ Source Code Translator

IcePick 3.0™
Original Award Winning MacApp 2.x and 3.x View Editor

Sierra Consulting Group
Top-Notch DataBase and SalesAutomation Consulting

Please call 800.621.0631 for a complete information package on Sierra's line of prestigious products and custom services.

Better Software through Innovation

SIERRA
SOFTWARE
INNOVATIONS

923 Tahoe Blvd., Incline Village, Nevada 89451
Phone: 702.832.0300 Fax: 702.832.7753 Internet: D2086@applelink.apple.com